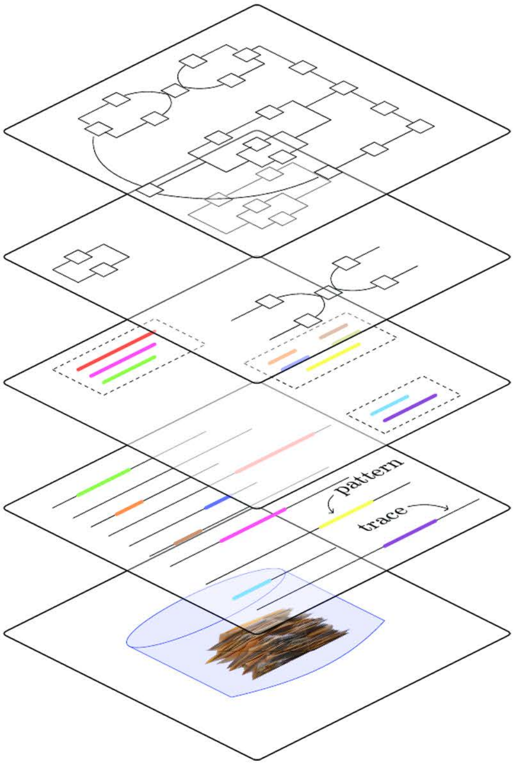
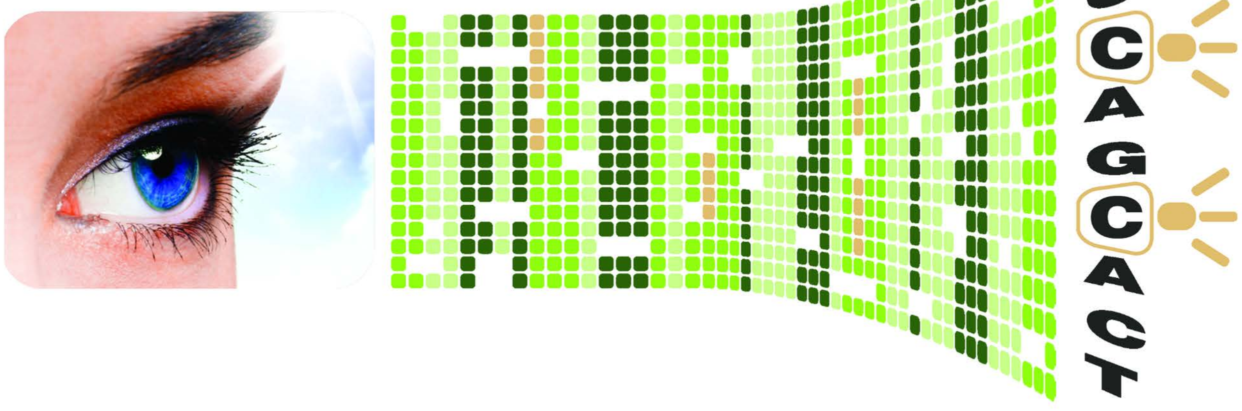
Process Mining in the Large
Preprocessing, Discovery, and Diagnostics

R.P. Jagadeesh Chandra Bose





Process Mining in the Large

Preprocessing, Discovery, and Diagnostics

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Jagadeesh Chandra Bose, R.P

Process Mining in the Large: Preprocessing, Discovery, and Diagnostics / by Jagadeesh Chandra Bose, R.P.
- Eindhoven: Technische Universiteit Eindhoven, 2012. - Proefschrift. -

A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-90-386-3118-9
NUR: 980
Keywords: Process Mining / Big Data / Process Discovery / Process Diagnostics / Concept Drift / Trace Clustering / Process Maps

The research described in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

SIKS Dissertation Series No. 2012–11

This work has been carried out as part of the “Process Mining within PMS” project, which has been sponsored by Philips Healthcare, The Netherlands.

Cover design: R.P. Jagadeesh Chandra Bose

Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 14 mei 2012 om 16.00 uur

door

Jagadeesh Chandra Bose Rantham Prabhakara

geboren te Chittoor, India

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. W.M.P. van der Aalst

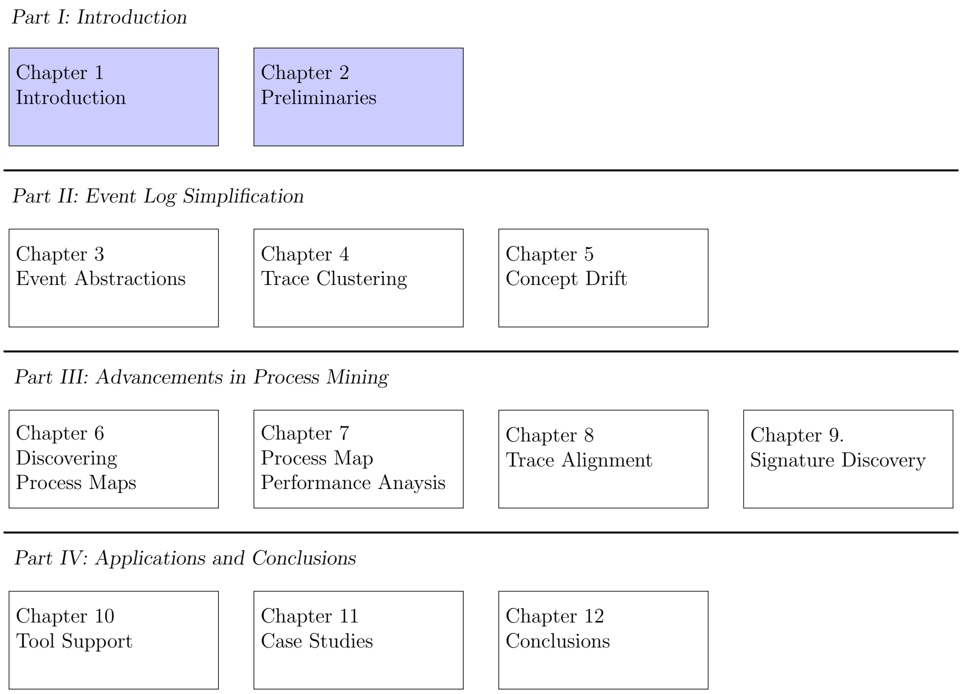
Copromotor:
dr.ir. B.F. van Dongen

To my parents

R. Prabhakara Naidu and R.P. Vimala Devi

Contents

Part I
Introduction



Process mining, which aims at extracting process-related information from event logs, has become an important tool for modern organizations that need to manage non-trivial operational processes. Today, we see an exponential growth of available data that can be used for analysis purposes. The volume of data and the complexity of processes creates many challenges for process mining. Chapter 1 highlights the motivating factors behind the research presented in this thesis and presents a unifying view of the contributions made in this thesis. Chapter 2 presents the basic concepts and notations that are needed for the rest of the thesis.

Chapter 1
Introduction

Process mining has made significant progress in less than a decade since its inception. Process mining techniques attempt to extract non-trivial process related knowledge and interesting insights from event logs. Process mining was welcomed with skepticism on its applicability in real-world situations with questions raised regarding the availability of event logs (data). The strides in technology over the last decade had turned the tables around with exponential growths in data, e.g., the decreasing costs of storage has fueled the creation of more and more information. Figure 1.1 depicts the projection of storage costs and the growth of data over the next decade. Many of today's information systems are recording an abundance of event logs. Process mining is now perceived as a critical link in the Business Process Management (BPM) life-cycle, and has been adopted in various commercial BPM systems (BPM|one, Futura Reflect, ARIS PPM, Fujitsu, Interstage, Businesscape, Iontas PDF, QPR PA, etc.).

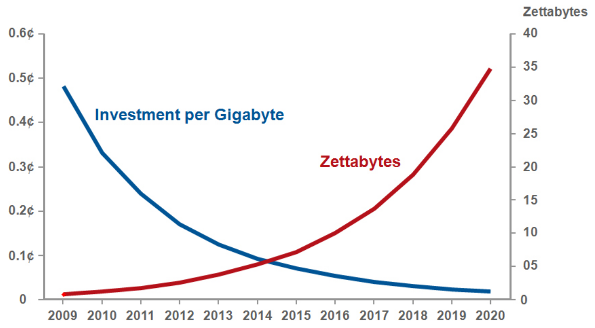


Figure 1.1: The decreasing cost of managing information is perceived to be an incentive to create more information. Source: IDC Digital Universe Study, Sponsored by EMC, May 2010 [79].

Today, we see an unprecedented growth of data from a wide variety of sources and systems across many domains and applications. The opportunities of data collection is going beyond traditional information systems to also include sensor and machine data. High-tech systems such as X-ray machines, vehicles, factory automation systems, and other devices are routinely instrumented to generate streams of data on their activities. For example, Boeing jet engines can produce 10 terabytes (TB) of operational information for every 30 minutes they turn. In just one Atlantic crossing, a four-engine jumbo jet can generate 640 terabytes of data [185]. Such voluminous data is emanating from many areas such as banking, insurance, finance, retail, healthcare, and telecommunications. For example, Walmart is logging one million customer transactions per hour and feeding information into databases estimated at 2.5 petabytes in size [185], a global Customer Relationship Management (CRM) company is handling around 10 million calls a day with 10–12 customer interaction events associated with each call [165]. The term "big data" has emerged to describe this phenomenon of data growth [147]. Organizations are raring to achieve new levels of insights from this data so that they can optimize and reinvent their business processes and provide better services to their customers. The complexity of data demands powerful tools to mine useful information and discover hidden knowledge. Process mining has become all the more relevant in this era of "big data" than ever before.

We are at the cross roads of an increasing number of domains and new applications willing to apply and adopt process mining. Process mining is being looked at even in applications that are atypical of workflow systems. Analysis of event logs of high-tech systems such as X-ray machines and CT scanners (medical systems), copiers and printers, mission-critical information systems, etc., are a few examples illustrating this trend. To give a specific example, Philips Healthcare has enabled the monitoring of its medical equipment (X-ray machines, MR machines, CT scanners, etc.) across the globe. Each of these systems records event logs capturing the operational events during its usage. Philips is interested in analyzing these event logs to understand the needs of their customers, identify typical use case scenarios (to test their systems under realistic circumstances), diagnose problems, service systems remotely, detect system deterioration, and learn from recurring problems.

Remarkable success stories have been reported on the applicability of process mining based on event logs from real-life workflow management/information systems. While these successes are certainly convincing, replicating this imposes certain requirements on the quality of event logs and the nature of processes to which one can apply process mining to. For example, *contemporary process discovery algorithms have problems in dealing with fine-grained event logs and less structured processes*, and generate *spaghetti-like process models* that are hard to comprehend [221]. They work well on high-quality event logs containing only events of interest with decently structured processes executed in a controlled environment, e.g., the event logs of a workflow management system. This is clearly in stark contrast to the characteristics of event logs and processes of high-tech systems. Event logs from high-tech systems are huge and tend to be fine-granular, heterogenous, and voluminous (cf. Section 1.2). For example, each Cardio-Vascular (CV) X-ray machine of Philips Healthcare logs around 350 KB of event data in compressed format (5 MB in uncompressed format) every day. Currently Philips has event logs from 2500 systems installed across the globe. This implies that Philips stores around 875 MB of compressed data every day. Figure 1.2(a) depicts the projection of cumulative data growth[1] even with a very conservative assumption of 3% growth in the number of CV systems every year. However, in reality, the complexity of systems increases steadily (owing to

[1]In reality, Philips may not be interested in logs older than 5 years for most of the analysis.

new requirements and added functionalities). This leads to more events and more data being logged. Figure 1.2(b) depicts the projection of cumulative data growth assuming a 10% increase in the number of events logged every two years. These figures illustrate the sheer volume of data that process mining techniques need to cope with and still produce meaningful results.

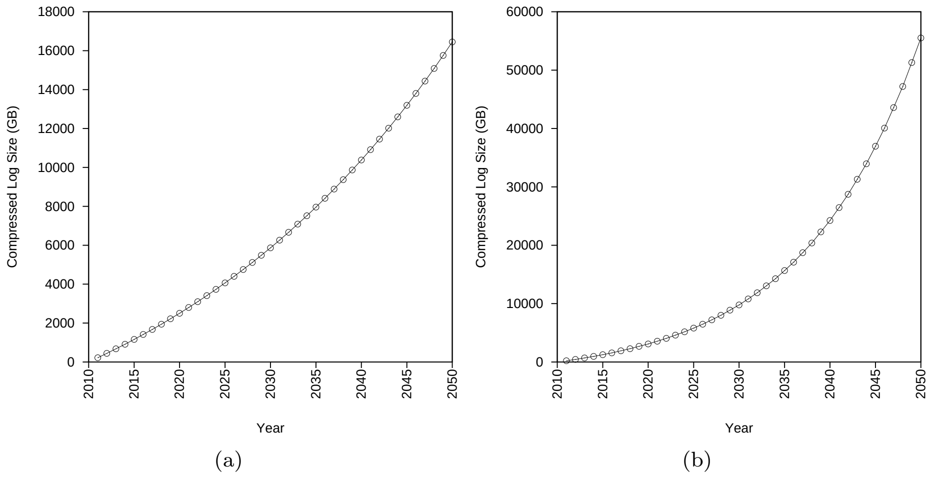


Figure 1.2: Projection of growth of data from cardio-vascular X-ray machines. (a) cumulative data growth with a 3% increase in the number of systems deployed every year (b) cumulative data growth with a 3% increase in the number of systems deployed every year and a 10% increase in the number of events logged every two years.

In addition to the characteristics of event logs, processes executed on high-tech systems tend to be flexible and/or less-structured, e.g., application of a medical procedure for a patient on an X-ray machine. *This calls for additional techniques and approaches to augment the repertoire of process mining techniques. Therefore, this thesis focusses on enabling process mining for large scale event logs.*

This chapter starts with an overview of process mining (Section 1.1). Then, the characteristics of event logs from high-tech systems are discussed (Section 1.2) followed by the three main challenges in process mining that we address in this thesis (Section 1.3). Finally, the chapter concludes with an overview of the contributions (Section 1.4) and the structure of this thesis (Section 1.5).

1.1 Process Mining

Process mining serves as a bridge between data mining and business process modeling. The goal of process mining is to extract *process-related knowledge* from event data recorded by a variety of systems (ranging from sensor networks to enterprise information systems). The starting point for process mining is the concept of an *event*. An event refers to a process instance, sometimes referred to as a case. An event e may have different properties, e.g., a *timestamp* (e occurred on January 1st 2011 at 10:00:20 EST), *resource information* (e was executed by John), *activity* (e corresponds to fluoroscopy procedure), *transaction type* (e is a start event, i.e., the start of the fluoroscopy procedure by John), and various *data elements* (e is done for a patient who is undergoing treatment for left coronary with a prescribed dosage of 1.25 units). All of these properties are optional. The only requirement is that events are *ordered* (i.e., no explicit timestamp is needed) and that each event belongs to a particular *class* (e.g., an activity name). Each process instance is described by a sequence of events referred to as a *trace*. If we use activity names as a classifier, then the trace corresponds to a sequence of activities. An event log is a multi-set of traces, i.e., a collection of traces where same traces may appear multiple times.

Figure 1.3 depicts an overview of process mining. The topics in process mining can be broadly classified into three categories (i) *discovery*, (ii) *conformance*, and (iii) *enhancement*. Process discovery deals with the discovery of models from event logs. These models may describe control-flow, organizational aspects, time aspects, etc. Today there are dozens of process discovery techniques generating process models using different notations (Petri nets, EPCs, BPMN, heuristic nets, etc.) [221]. Figure 1.4 illustrates the basic idea of process discovery. An event log containing detailed information about events is transformed into a multiset of traces $\mathcal{L} = [\texttt{abcdjkln}, \texttt{aefjkmn}, \texttt{abgchdjkln}, \texttt{agefjkmin}, \ldots]$. Process discovery techniques are able to discover process models such as the Petri net shown in Figure 1.4.

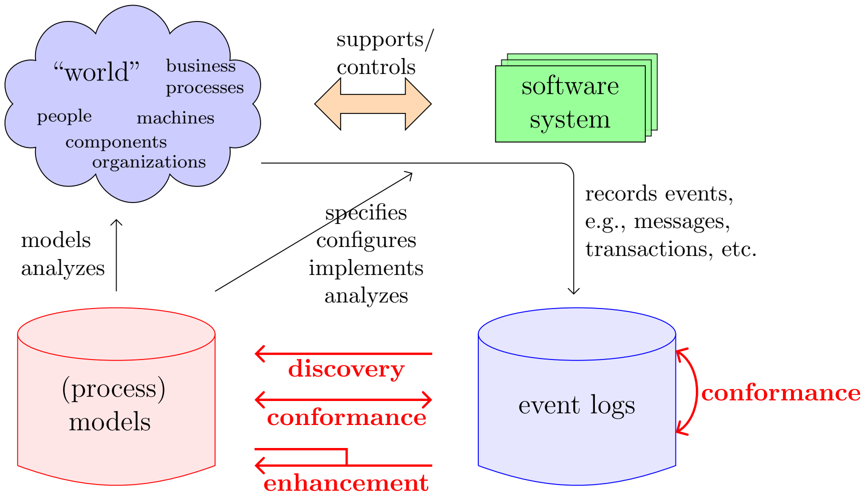


Figure 1.3: An overview of process mining (adapted from [221]). The thicker arcs indicate the positioning of the three main topics in process mining, viz., *discovery*, *conformance*, and *enhancement*.

Conformance deals with comparing an apriori model with the observed behavior as recorded in the log and aims at detecting inconsistencies/deviations between a process model and its corresponding execution log. In other words, it checks for any violation between *what was expected to happen* and *what actually happened*.

case id	event id	properties				
		time stamp	activity	resource	cost	...
	10001001	01-10-2010:09:32	register	Bob	10	...
	10001002	02-10-2010:11:17	high insurance check	Alice	15	...
	10001003	02-10-2010:16:43	high medical history check	Alice	15	...
1	10001004	03-10-2010:13:54	contact hospital	Alice	20	...
	10001005	04-10-2010:18:32	decide	Wil	150	...
	10001006	05-10-2010:09:05	prepare notification	Bob	10	...
	10001007	05-10-2010:10:13	send notification by email	Bob	10	...
	10001008	05-10-2010:10:44	archive	Bob	10	...
	10002001	01-10-2010:11:01	register	Anita	10	...
	10002002	04-10-2010:14:23	low insurance check	Anu	12	...
	10002003	05-10-2010:10:37	low medical history check	Anu	12	...
2	10002004	08-10-2010:08:16	decide	JC	50	...
	10002005	10-10-2010:14:05	prepare notification	Anita	10	...
	10002006	11-10-2010:15:13	send notification by post	Anita	10	...
	10002007	14-10-2010:09:41	archive	Anita	10	...
	10001231	11-11-2010:10:32	register	Mike	11	...
	10001232	12-11-2010:12:13	high insurance check	Kate	17	...
	10001233	12-11-2010:13:14	send questionnaire	Mike	23	...
	10001234	22-11-2010:14:23	high medical history check	Kate	17	...
	10001235	22-11-2010:15:41	receive response	Sara	17	...
3	10001236	03-12-2010:06:54	contact hospital	Peter	23	...
	10001237	04-12-2010:08:12	decide	Chase	100	...
	10001238	15-12-2010:13:05	prepare notification	Sasha	10	...
	10001239	15-12-2010:17:13	send notification by email	Sasha	10	...
	10001240	17-12-2010:18:14	archive	Mike	10	...
	10006711	03-01-2011:13:01	register	Tom	10	...
	10006712	04-01-2011:11:33	send questionnaire	Harry	12	...
	10006713	06-01-2011:09:43	low insurance check	Mika	12	...
	10006714	15-01-2011:11:17	low medical history check	Mika	12	...
4	10006715	18-01-2011:19:16	decide	Mark	80	...
	10006716	20-01-2011:17:05	prepare notification	Dash	10	...
	10006717	31-01-2011:05:13	send notification by post	Kim	10	...
	10006718	04-02-2011:17:23	skip response	Tom	12	...
	10006719	14-02-2011:09:41	archive	Kate	10	...
...	...	...	...	...	...	...

`a` - register
`b`-high insurance check
`c`-high medical history check
`d`-contact hospital
`e`-low insurance check
`f`-low medical history check
`g`-send questionnaire
`h`-receive response
`i`-skip response
`j`-decide
`k`-prepare notification
`l`-send notification by email
`m`-send notification by post
`n`-archive

```
a b c d j k l n
a e f j k m n
a b g c h d j k l n
a g e f j k m i n
...
```

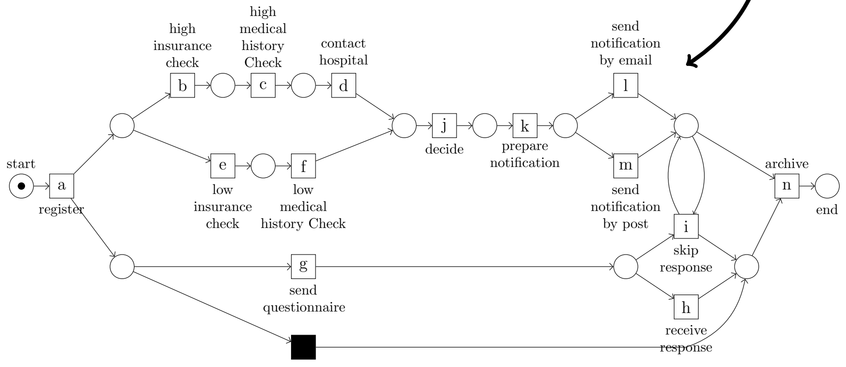


Figure 1.4: Process discovery aims to learn a process model (in this case, a Petri net) from traces of activities.

Enhancement deals with extending or improving an existing model based on information about the process execution in an event log. For example, annotating a process model with performance data to show bottlenecks, throughput times, etc., by exploiting the timestamps in the event log.

1.2 Characteristics of High-Tech System Event Logs

In this section, we discuss some of the characteristics of event data from high-tech systems.

- *Event granularity:* Event logs from high-tech systems are very fine grained and too detailed for most stakeholders. The granularity at which events are logged varies widely (across domains/applications) without any specific consideration for the contexts of analysis. More often than not, the events in an event log are at different levels of granularity. This problem is compounded by the lack of any standard/guideline on logging specifications. Viewing the process at the right abstraction level is important for the end-user/analyst.
- *Case heterogeneity:* High-tech systems are complex large scale systems supporting a wide range of functionality. For example, medical systems support medical procedures that may have hundreds of potential variations. Furthermore, these systems are typically designed to be quite *flexible* in their operation. This results in event logs containing a heterogeneous mix of usage scenarios with more diverse and less structured behavior. Although it is desirable to also record the exact variant of the use case, it is often not available, and in some cases, it is infeasible to define all the variants. Another source of heterogeneity stems from operational processes that change over time to adapt to changing circumstances, e.g., new legislation, extreme variations in supply and demand, seasonal effects, etc. For example, a new medical regulation might force the medical equipment manufacturers to alter their machines. This might have an influence in the way a particular procedure is applied. As another example, an experienced physician who has been newly recruited in a hospital might apply a medical procedure in a more efficient manner. One can notice a difference in the way the same procedure had been applied before and after the arrival of this physician. Diversity in an event log can also be attributed to such changes.
- *Voluminous data:* High-tech systems produce large amounts of data, because they typically capture very low-level events such as the events executed by the system components, application level events, network or communication events, and sensor data (indicating status of components etc.). Each atomic event in these environments has a short life-time and hundreds of events can be triggered within a short time span (even within a second). This poses the challenge on the availability and scalability of analysis techniques to cope with such voluminous data.
- *Unreliable timestamps:* The ordering of events from high-tech system logs may not always be reliable. For example, an X-ray machine has dozens of components with each component having a local clock and a local buffer. There could be a mismatch between the times when an event is actually triggered and when an event is recorded (an event is first queued in the internal buffer of a component before it is logged). Moreover, the recording of events across different components is done in a distributed manner. Furthermore, discrepancies in timestamps can also occur in cases where the clocks across different components

are not synchronized. Such discrepancies add to the complexity of analysis and more importantly can lead to incorrect insights if not handled properly. For example, *cause* and *effect* may be swapped because of the incorrect ordering of events.

- *Scoping:* Another issue is concerned with defining what constitutes a case and in being able to generate/identify a case from the data sources. The definition of a case can have different connotations based on the contexts and purpose of analysis. There are two modes in which event data is typically made available: (i) *centralized or monolithic* mode, where a single log captures every thing related to a particular system on a single day, and (ii) *distributed* mode where the data related to a system is stored across *disparate data sources*. Appropriate *scoping* is essential to discover correct insights or to be able to answer questions. For example, in an X-ray machine event data, completely different aspects need to be considered when it comes to gaining insights on (i) the real usage of the system and (ii) recurring problems and system diagnosis, the former requiring the analysis of commands/functions invoked on the system while the latter needing error and warning events. Domain knowledge is needed in defining such an appropriate scope.

There are two important points to bear in mind with respect to the above list. Firstly, the above mentioned aspects are neither exhaustive nor orthogonal. There exists considerable overlap between some the items, e.g., fine granularity of events is one of the contributing factors to voluminous data. Secondly, though the problems mentioned above are more pronounced in high-tech system event logs, it is not uncommon to notice these even in event logs of other systems, e.g., traditional enterprise information systems [147]. Henceforth, we refer to logs with the above characteristics as large scale event logs.

1.3 Challenges in Process Mining

Existing process mining techniques have shown their applicability in workflow-like processes [221]. However, analyzing event logs from less structured processes as seen in the context of high-tech systems is more difficult. Recently, the IEEE Task Force on Process Mining has articulated a list of challenges in process mining in its manifesto [167]. In this thesis, we address the following three challenges, which are related to the characteristics mentioned in Section 1.2:

- *Dealing with less-structured processes:* most processes mined from real-life logs tend to be less structured than what stakeholders expect. The discovered process models are often spaghetti-like and are hard to comprehend. Many factors lead to such a behavior. The *heterogeneity of cases* and *fine granular events* are two of the primary factors. Another dimension to this problem stems from the limitations of the existing process discovery algorithms. The majority of process discovery techniques in the literature pertain to the discovery of control-flow models that are "flat" [229, 238, 246, 264]. A notable exception is the Fuzzy miner [94]. Flat models have inherent limitations and are one of the primary sources of spaghettiness, especially when it comes to dealing with large scale logs and less structured and flexible processes. For a log with $|\mathcal{A}|$ event classes (activities), a flat model can be viewed as a graph containing $|\mathcal{A}|$ nodes with edges corresponding to the causality defined by the execution behavior in the log. Graphs become quickly overwhelming and unsuitable for human perception and cognitive systems even if there are just a few dozens of nodes [83]. This problem is compounded if the graph is dense (which is often the case in unstructured processes) thereby compromising the comprehensibility of models. Figure 1.5 shows an example of a typical spaghetti process discovered using conventional process mining techniques [221]. The complexity of the diagram illustrates the problems and challenges.

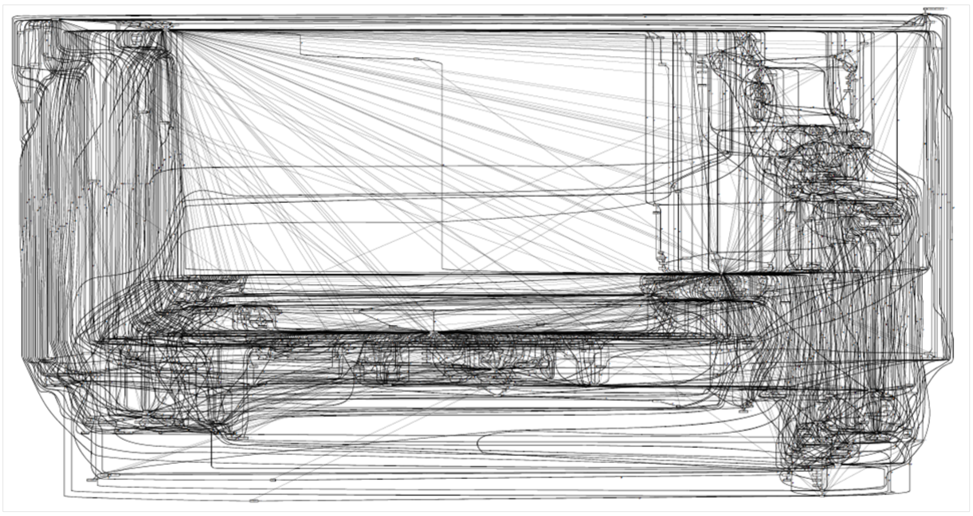


Figure 1.5: Spaghetti process describing the diagnosis and treatment of 1143 patients of the Gynaecology department in a Dutch hospital. The process model was constructed based on an event log containing 150, 291 events. There are 624 different activities (taking event types into account). This event log is provided for the first business process intelligence challenge `doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`.

Process models can be seen as "maps" describing the operational processes. *There is a need for techniques that enable the discovery of navigable process maps with seamless zoom-in/zoom-out facility (i.e., hierarchical process models with different perspectives).*

- *Dealing with process changes:* Contemporary process mining techniques assume the processes to be in steady state. For example, when discovering a process model from event logs, it is assumed that the process at the beginning of the recorded period is the same as the process at the end of the recorded period. However, as mentioned earlier, processes may change to adapt to changing circumstances, e.g., new legislation, extreme variations in supply and demand, seasonal effects, etc. *Concept drift* refers to the situation in which the process is changing while being analyzed [22]. *There is a need for techniques that deal with such "second order dynamics".* Analyzing such changes is of utmost importance to get an accurate insight on process executions at any instant of time.

- *Provisions for process diagnostics:* lion's share of process mining research has been devoted to control-flow discovery. Process diagnostics, which encompasses process conformance checking, auditing, process performance analysis, anomaly detection, diagnosis, inspection of interesting patterns and the like, is gaining prominence in recent years [3, 4, 15, 26, 191, 217, 222, 230, 235, 237, 241, 271]. Organizations are interested in uncovering answers to a variety of diagnostic questions such as: are there any bottlenecks in the process?, where do process instances deviate?, are there any common patterns of execution in the process instances?, are there any symptomatic patterns that lead to a failure?, who typically work together?, etc. Techniques such as conformance checking [191] and process performance analysis [241] have inherent limitations. They assume the availability of process models. However, in reality, models are either not present or even if they are present they are not up-to-date or too complex (as in Figure 1.5). Looking inside of such processes to find answers is *implausible.* Techniques such as LTL checker [230] can assist in addressing this problem only to a certain extent (they are not suitable for explorative analysis) and do not scale up to dealing with large scale event logs. Therefore, *there is a need for complementary techniques to assist auditors and analysts in their diagnostic efforts.*

1.4 Contributions of this Thesis

In this thesis, we hypothesize that the problems arising in analyzing large scale event logs can be attacked from two fronts: (i) *event log simplification* and (ii) *advancements in process mining.* The following subsections provide a high-level summary on these two aspects.

1.4.1 Event Log Simplification

We propose that some of the problems highlighted earlier in applying process mining on large scale event logs can be addressed through systematic pre-processing in order to simplify the event log (scope, filter, etc.).

Dealing with fine-granular event logs

In an event log, there can be scenarios where the system is subjected to similar execution patterns and behavior within and across cases. Figure 1.6 depicts an example showing common execution patterns among the traces. Such commonalities are not without a reason and they typically carry a strong conceptual relationship (of domain significance) between the events involved in the behavior. We exploit some of the sequence patterns, such as *tandem arrays* and *maximal repeats* familiar in the string-processing and bioinformatics literature [97, 98, 130, 131] and adapt them to the process mining domain. We establish a relation between the manifestation of some of the process model constructs and the sequence patterns. Furthermore, we define various metrics that assist in assessing the significance of the patterns uncovered and propose a means to form abstractions over these patterns. The abstractions thus formed can act as high-level activities representing the low-level events captured underneath it. Event logs can be simplified by replacing the low-level events with these abstract activities thereby mitigating the problem with event granularity.

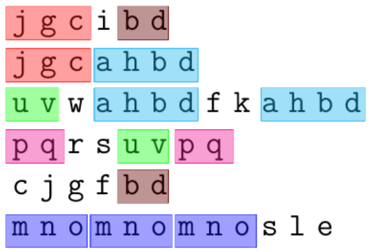


Figure 1.6: Common execution patterns within and across traces in an event log.

Dealing with heterogeneity in event logs

Heterogeneity in event logs can be dealt with trace clustering, which pertains to the *grouping together of homogenous sets of cases.* Figure 1.7(a) depicts a heterogenous mix of traces (distinguished by color). The objective of trace clustering is to partition the traces into clusters such that traces within a cluster are similar to each other and traces that belong to different clusters are dissimilar. For the above example, this implies grouping together of traces of the same color as illustrated in Figures 1.7(b)-(f). Trace clustering has been reported to be an effective approach when dealing with diversity in an event log resulting from less-structured and flexible processes. It has been argued that process mining results can be improved by partitioning an event log into subsets of homogenous cases and analyzing these subsets independently [54, 86, 87, 89, 209]. Figure 1.8 illustrates the significance of trace clustering in process mining. The process model on the top of Figure 1.8 is a process model mined from an entire event log. This model is quite complex to comprehend. The bottom rectangles of Figure 1.8 depict the process models mined from the clustered traces. It is evident that clustering enables the comprehension of process models by reducing the spaghettiness.

Most of the arguments on ‘improved results’ through trace clustering, though convincing, are subjective in nature. There are three factors that influence the partitioning of an event log into clusters of homogenous cases: (i) the features used to characterize a case, (ii) the distance or similarity metrics used, and (iii) the choice of clustering algorithm (and its parameters). Interpretation of the formed clusters is always relative to the choices made for each of the three factors. In this thesis, we highlight some of these issues. We analyze feature selection and distance (or similarity) metrics in contemporary approaches to trace clustering and propose new feature sets that are process-centric. Process-centric feature sets enable the interpretation of the formed clusters from a process perspective. In addition, we propose

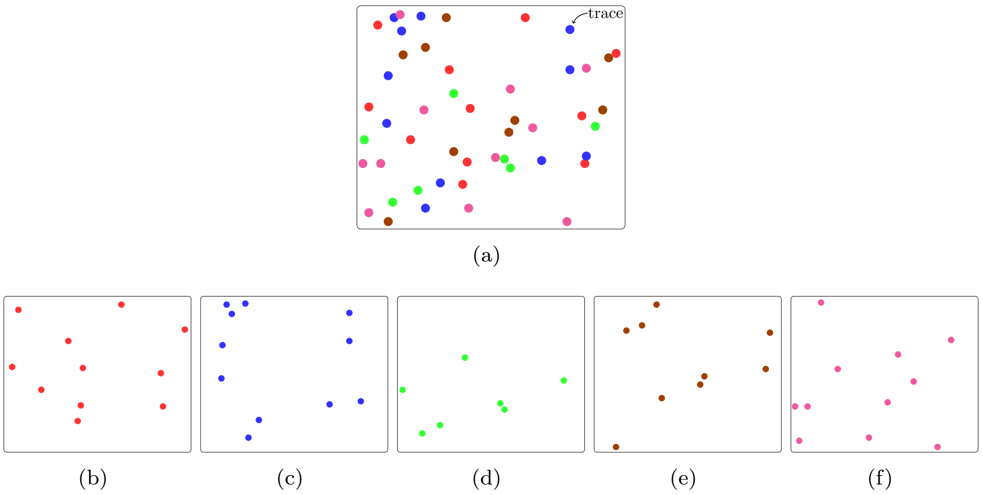


Figure 1.7: Objective of trace clustering: (a) an event log containing a heterogenous mix of traces. Traces represented by the same color are similar to each other and traces of different colors are dissimilar, (b) segregating all red traces, (c) segregating all blue traces, (d) segregating all green traces, (e) segregating all brown traces, and (f) segregating all magenta traces.

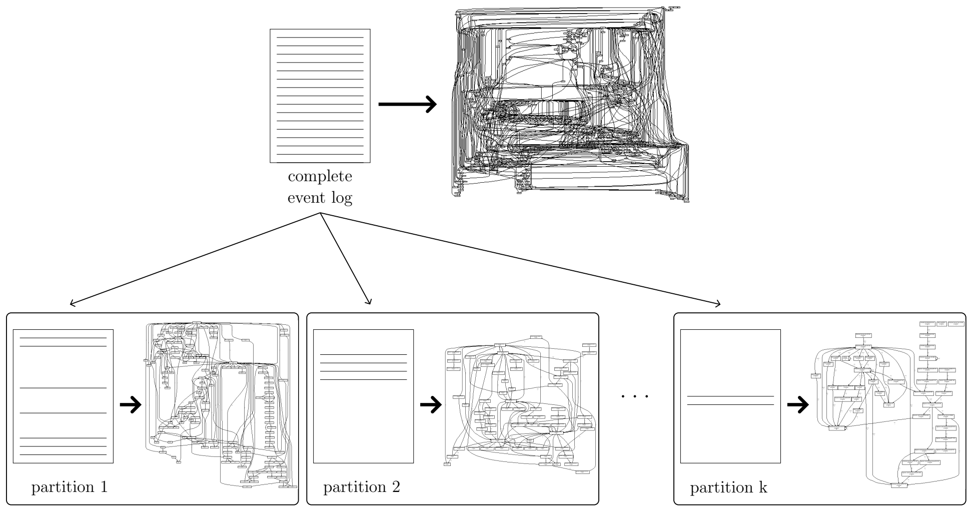


Figure 1.8: Significance of trace clustering in process mining.

a context-aware approach to trace clustering that alleviates the need for defining features and feature selection. Furthermore, we propose objective metrics to compare and assess the goodness of the formed clusters from a process mining point of view.

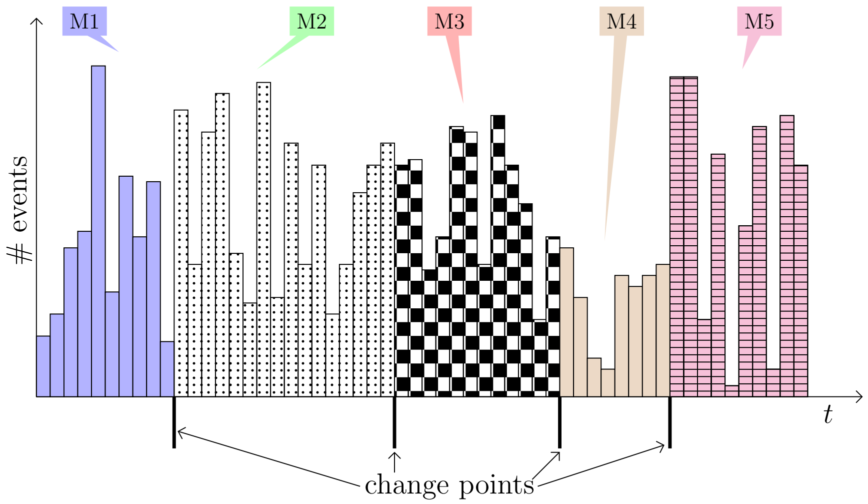


Figure 1.9: An event log containing traces from different process variants along with the points of change.

Dealing with process changes

As mentioned earlier, it is not uncommon for operational processes to adapt to changing circumstances. Although flexibility and change have been studied in-depth in the context of WFM and BPM systems, contemporary process mining techniques assume the process to be in steady state. Obviously, this is often not the case due to the phenomenon known as *concept drift*. An idealistic requirement is to record the sequence of process changes performed, in the form of a change log, so that event logs can be analyzed appropriately [95]. However, such a change log will not be available in most cases. As a result, process changes manifest themselves only *latently* in the event logs (in the way which activities are executed when, how, and by whom). In this thesis, we introduce various facets of concept drift and propose approaches to detect drifts. Figure 1.9 depicts an event log containing cases from a process that has undergone four changes and the points when those changes have taken place. One of the objectives of handling concept drifts in process mining is to be able to detect when the processes have changed. Once the change points are uncovered, one can consider the cases between the points of change as belonging to one concept and those across the points to be diverse. In other words, detecting drifts addresses the issue of case heterogeneity and enables the *grouping of cases*. Trace clustering and handling concept drifts are complementary approaches and can be applied together to exploit their advantages.

1.4.2 Advancements in Process Mining

The above three topics, viz., abstractions of events, trace clustering, and handling concept drifts, mitigate some of the issues identified in dealing with large scale event logs through log simplification. Though log simplification is a critical step, it alone is not sufficient to address the questions with which process mining is sought after as a means to provide answers to. Inability to provide answers is primarily either due to the unavailability of an appropriate technique or due to the limitations with existing techniques in process mining. As process mining is a relatively young field, several challenging problems remain to be addressed [167, 221]. In this thesis, we take a step forward towards this and propose techniques for dealing with less-structured processes and for assisting analysts in process diagnostics.

Dealing with less-structured processes

We highlighted the issues with current techniques in process discovery in Figure 1.5. Process modelers and business analysts typically attack large, complex, and unstructured processes using hierarchies. Hierarchical process modeling is a modeling paradigm where a process is modeled on multitude levels of detail, such that lower levels or subprocesses are included in higher level processes. However, attempts at process simplification in process mining are mostly confined to techniques that retain highly significant information while discarding less significant ones [175]. A notable exception is the Fuzzy miner [94]. Taking *cartography* as a metaphor, Fuzzy miner attempts to provide a hierarchical model, but limited to one level of hierarchy. Less significant activities and edges are either removed or clustered together in the model. The cluster can be zoomed-in to uncover the abstraction captured underneath it. However, Fuzzy miner poses a danger of clustering activities/edges having no domain significance. One can imagine this to a scenario where some streets in Amsterdam are combined with streets in Eindhoven in a map.

In this thesis, *we provide a new dimension to hierarchical process discovery.* As mentioned earlier, process models can be seen as the "maps" describing the operational processes of organizations. Analogous to cartography, process models should allow for various *context-dependent views*. Typically, different stakeholders would be involved in a process and only some of them would be interested in knowing the entire process in detail. Each stakeholder would be more interested to see the portions of the process falling in the realm of his/her area of concern or responsibility in great detail and the rest of the process in abstract notions. We propose a two-phase approach to process map discovery. In the first phase, called the pre-processing phase, we exploit the concepts proposed in dealing with fine-granular event logs, i.e., abstractions of events, and select those activities and abstractions that are relevant in the context of analysis. We transform the original event log using these abstractions into an abstract event log and create a sub-log for each abstraction. The sub-log of an abstraction captures the sub-traces corresponding to the abstraction. In the second phase, a process model is mined from the abstract event log. The node corresponding to an abstraction in this model can be seamlessly zoomed-in to reveal the subprocess captured by it. Multiple levels of hierarchy can be achieved by employing the pre-processing phase repeatedly.

Figure 1.10 highlights the difference between the traditional approach to process discovery and the two-phase approach. In the two-phase approach, we first identify common execution patterns in the event log and define abstractions over them, e.g., the activity subsequence `sam` is represented by the abstract activity `X`, the activity subsequences `cudn`, `cdnu`, and `dcnu` are captured by the abstract activity `Z`. The event log is then transformed using these abstractions and a process model is mined on the transformed log. The abstract activities can be zoomed in to see the subprocesses captured underneath them as depicted in the figure. Note that the process model (map) mined using the two-phase approach is simpler. The exploitation of common execution patterns and its relationship to process discovery can be summarized as depicted in Figure 1.11.

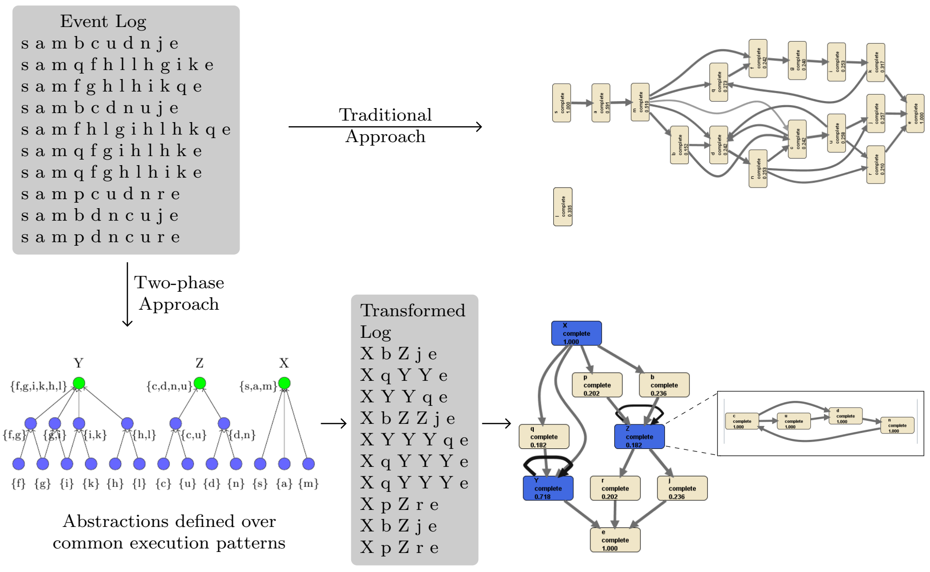

```
Event Log
s a m b c u d n j e
s a m q f h l l h g i k e
s a m f g h l h i k q e
s a m b c d n u j e
s a m f h l g i h l h k q e
s a m q f g i h l h k e
s a m q f g h l h i k e
s a m p c u d n r e
s a m b d n c u j e
s a m p d n c u r e
```
```
Transformed
Log
X b Z j e
X q Y Y e
X Y Y q e
X b Z Z j e
X Y Y Y q e
X q Y Y Y e
X q Y Y Y e
X p Z r e
X b Z j e
X p Z r e
```
Traditional Approach
Two-phase Approach
Y
Z
X
{f,g,i,k,h,l}
{c,d,n,u}
{s,a,m}
{f,g}
{g,i}
{i,k}
{h,l}
{c,u}
{d,n}
{f} {g} {i} {k} {h} {l} {c} {u} {d} {n} {s} {a} {m}
Abstractions defined over common execution patterns

Figure 1.10: Traditional approach versus our two-phase approach to process discovery. The two-phase approach enables the mining of hierarchical process models by defining abstractions. The blue-colored (dark) nodes in the process model depict abstract activities, which can be zoomed in to view the subprocesses captured by them.

Provisions for process diagnostics

As mentioned earlier, process diagnostics encompasses process conformance checking, auditing, process performance analysis, anomaly detection, diagnosis, inspection of interesting patterns and the like. Process discovery serves as the starting point of process mining. As shown in Figure 1.3, the discovered model can be combined with the event logs to extract further insights. Discovered process models reflect the "as-is" execution behavior of a process from a control-flow perspective. A comprehensive picture about the reality can be obtained by annotating the process models with additional information such as timing information and resource information. Furthermore, such enhancements are essential to assist in diagnostic efforts, e.g., to identify the bottlenecks in a process. These annotations can be obtained by replaying the event log on the model and gathering relevant statistics such as the

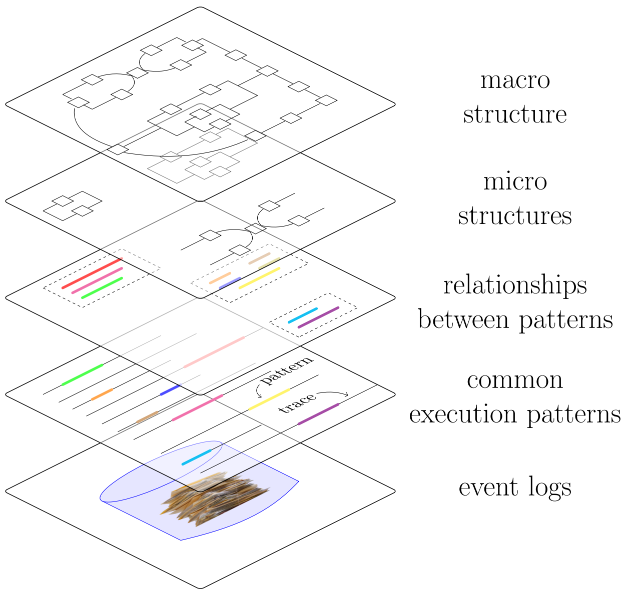


Figure 1.11: Repeating subsequences of activities define the common execution patterns and carry some domain (functional) significance. Related patterns and activities pertaining to these patterns define abstractions that correspond to micro-structures (or subprocesses). The top-level process model can be viewed as a macro-structure that subsumes the micro-structures.

number of executions and the average execution time. Contemporary approaches to performance analysis in process mining are confined to flat models [241] and cannot be applied to process maps that support multiple levels of hierarchy. In this thesis, we present a technique to replay an event log onto the process map and compute various KPIs (key performance indicators).

Process performance analysis based on replay (as mentioned above) and conformance checking (as discussed in Section 1.3) both require the availability of a process model. However, in reality, models are either not present or even if present are not up-to-date or can be too complex (as in Figure 1.5). Looking inside of such processes to find diagnostic information such as deviations and performance bottlenecks is implausible. There is a need for complementary techniques to assist auditors and analysts in their diagnostic efforts. Given an event log, we would like to answer a variety of diagnostic questions:

1. *What is the most common (likely) process behavior that is executed?* For a given event log, it would be interesting to know which process components are essential/critical for this process. Such essential components/functions form the backbone of the process and should be conserved. Process re-design/improvement efforts should focus on improving such critical components.
2. *Where do process instances deviate?* In practice, there is often a significant gap between what is prescribed or supposed to happen, and what actually happens.

There is a need to augment process diagnostics with techniques that can assist in finding deviations by analyzing raw traces in the event logs. There are many domains/applications where this requirement is felt, e.g., fault diagnosis, anomaly detection, etc. Given an event log containing a mix of traces where the system (process) functioned normally and where it malfunctioned, analyzing these traces to find deviations in malfunctioned/anomalous traces from normal traces would give cues in understanding the cause of malfunction/anomaly.

3. *Are there any common patterns of execution in the traces?* Event logs may contain data indicating the health of a process or the status of a case, etc. One can consider such health indicators as class labels. For example, an X-ray machine event log might contain information on system failures and broken parts/components; an insurance claim event log might contain information on whether a claim is fraudulent or not. Organizations are interested in gaining further insights on such health indicators such as learning whether there are any common patterns among the cases with a certain class label or whether there are any discriminatory patterns between cases of different classes. For example, understanding the contexts in which an X-ray machine breaks down, the symptoms before a failure, etc.

4. *What are the contexts in which an activity or a set of activities is executed in an event log?* Dependencies exist between activities in a process, and activities are expected to be executed within a certain context. There can be short-range and long-range dependencies between activities. Long-range dependencies are difficult to discover. An analyst is interested in understanding the contexts of execution of activities and/or activity sequences.

5. *What are the process instances that share/capture a desired behavior either exactly or approximately?* Often in diagnostics, an analyst is interested in finding process instances that share/comply to a particular desired behavior; the desired behavior can be expressed as a manifestation of some pattern of activity sequences or some complex form (combination) of these patterns. Although temporal logic approaches [142, 230] can assist in addressing this problem to a certain extent (by discovering process instances that capture the desired behavior exactly), one might also be interested in discovering process instances that share the desired behavior approximately.

6. *Are there particular patterns (e.g., milestones, concurrent activities, etc.) in the process?* Workflow patterns [228] refer to recurring forms/structures addressing business requirements. For example, milestones indicate specific execution points in the process model and provide a mechanism for supporting the conditional execution of a task or a subprocess. An analyst is interested in discovering the presence of, and in analyzing milestone patterns in the process event log. Similarly, discovery of process models with concurrency is one of the challenging problems in process mining. The presence of concurrent activities creates different permutations of activities in the event log that adds to the complexity of discovery algorithms. Detection of the presence of concurrent activities also helps in pre-processing the logs.

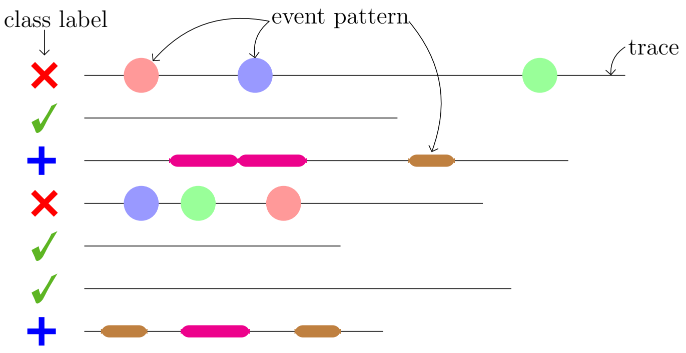


Figure 1.12: An example of finding signature patterns that can discriminate between different classes of behavior.

In this thesis, *we propose techniques to address the above diagnostic questions.* We model the problem of finding patterns that are common within a particular class or discriminatory across different classes as a classification problem in data mining. As a result, we discover symptomatic signature patterns in the form of rules. Figure 1.12 depicts a high-level view of this problem. There are three classes of behavior (indicated by the class label for each trace) and we are interested in finding patterns (if any) in traces that can distinguish between the three classes. We propose techniques that can uncover such patterns, e.g., the event patterns represented by circles are specific to traces labeled with the class ✖, the event patterns represented by bars are specific to traces labeled as ✚, etc. Such signature patterns can be used in applications such as fault diagnosis and prediction.

Furthermore, we hypothesize that if processes are less structured and event logs are far from complete, then it is better to carefully inspect the event log by grouping and aligning the traces found in the event log. We propose an approach called *trace alignment* whose goal is to align traces in such a way that event logs can be explored easily. Figure 1.13 depicts the result of trace alignment along with annotations indicating how trace alignment can assist in answering some of the diagnostic questions raised above. By aligning traces we can see the common and frequent behavior, and distinguish this from the exceptional behavior. For example, given an event log containing a mix of traces where the system process functioned normally and where it malfunctioned, aligning the traces can help in finding the deviations in malfunctioned traces when compared to normal traces. Such deviations give cues in understanding the cause of malfunction. Trace alignment complements existing process mining techniques focusing on discovery and conformance checking. It creates an altogether new dimension to conformance checking; *deviations and violations are uncovered by analyzing just the raw event traces* (thereby avoiding the need for process models). This is illustrated by Figure 1.3, which shows two types of conformance checking.

The concepts presented in this thesis have been published in leading journals and conferences/workshops (see Appendix B for the list of publications).

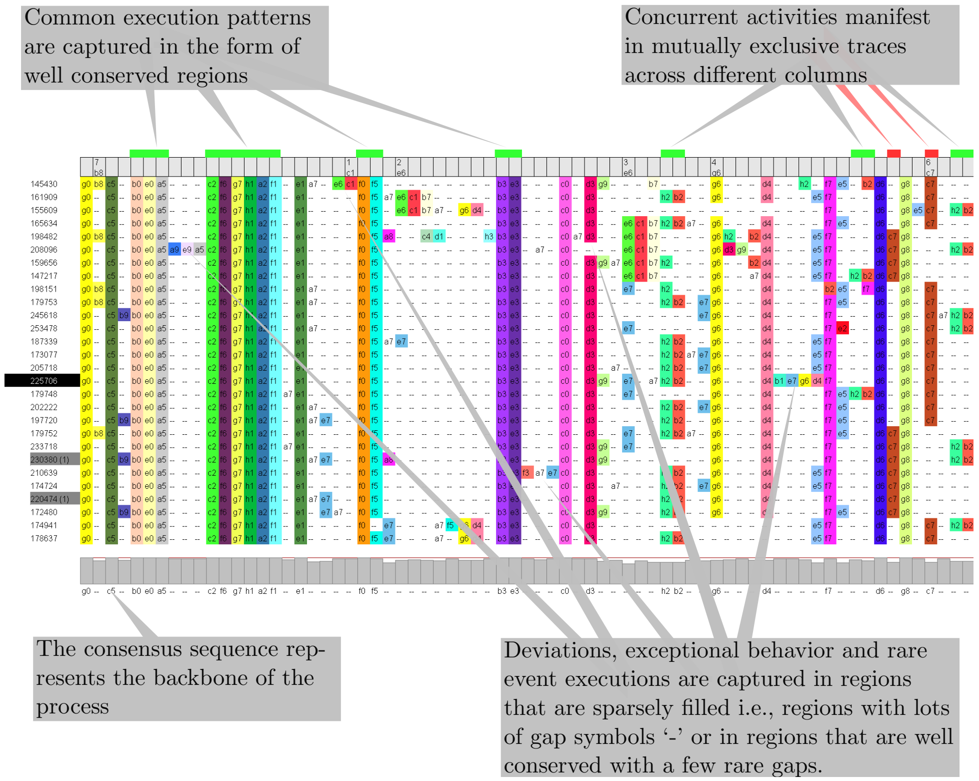

Common execution patterns are captured in the form of well conserved regions
Concurrent activities manifest in mutually exclusive traces across different columns
The consensus sequence represents the backbone of the process
Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lots of gap symbols '-' or in regions that are well conserved with a few rare gaps.

Figure 1.13: An example of trace alignment. Each row refers to a trace. Columns describe positions in traces. Consider now the cell in row y and column x. If the cell contains an activity name a, then a occurred for case y at position x. If the cell contains no activity name (i.e., a gap "-"), then nothing happened for y at position x.

1.5 Structure of this Thesis

This thesis is structured as depicted in Figure 1.14 and is divided into four main parts. Part I constitutes the introduction to the thesis, which includes the current chapter and concludes with the Preliminaries (Chapter 2) where we introduce the concepts and notations such as sets, graphs, sequences, process models, and event logs that are needed for this thesis. Part II deals with the systematic pre-processing leading to the simplification of event logs (Chapters 3–5). Chapter 3 on *abstractions of events* deals with the issue of fine-granular event logs. We present techniques to form abstractions by exploiting the common execution patterns in an event log. Chapter 4 is dedicated to *trace clustering* as a means of dealing with heterogeneity in event logs. We argue that considering contexts is important to forming good clusters and present context-aware approaches to trace clustering. We also present metrics to assess the goodness of clusters from a process mining point of view. Chapter 5 on *concept drift* caters to the analysis of process changes in event logs. We introduce the topics in handling concept drift in process mining and present techniques to detect change points in an event log.

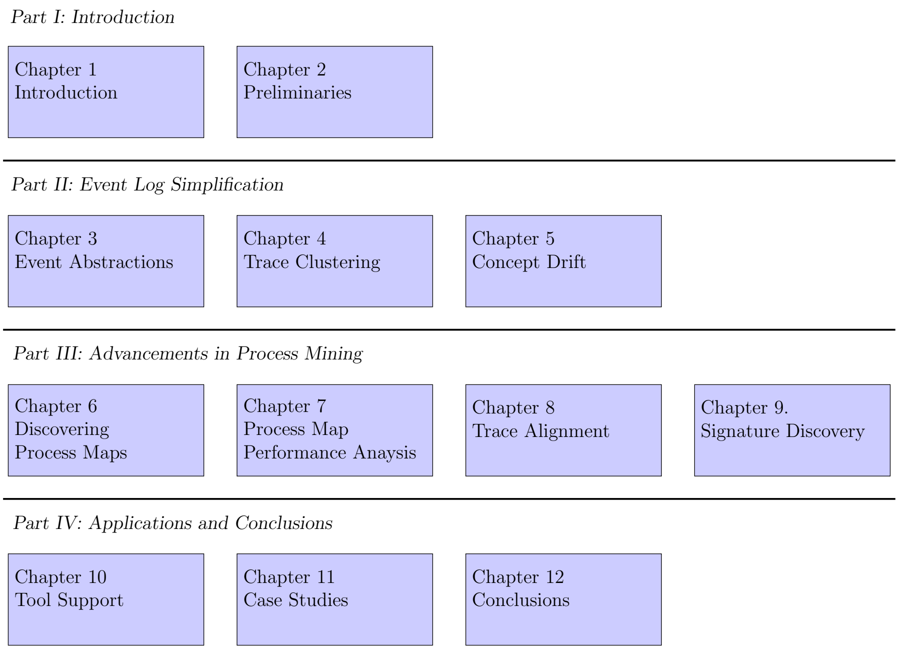

Part I: Introduction
Chapter 1 Introduction
Chapter 2 Preliminaries
Part II: Event Log Simplification
Chapter 3 Event Abstractions
Chapter 4 Trace Clustering
Chapter 5 Concept Drift
Part III: Advancements in Process Mining
Chapter 6 Discovering Process Maps
Chapter 7 Process Map Performance Anaysis
Chapter 8 Trace Alignment
Chapter 9. Signature Discovery
Part IV: Applications and Conclusions
Chapter 10 Tool Support
Chapter 11 Case Studies
Chapter 12 Conclusions

Figure 1.14: Structure of this thesis.

Part III of the thesis deals with the advancements in process mining (Chapters 6–9). It starts with Chapter 6 on *discovering process maps* via a two-phase approach to deal with less structured processes. This chapter utilizes the abstractions of low-level events defined in Chapter 3 and presents an event log transformation algorithm that replaces manifestations of common execution patterns with abstract activities. Chapters 7–9 focuses on techniques for process diagnostics. Chapter 7 on *process map performance analysis* deals with enriching the discovered process maps with performance metrics. This chapter presents a replay technique that facilitates the computation of key performance indicators and thereby assist in the identification of bottlenecks in a process. In Chapter 8, we present an approach to align traces in such a way that event logs can be explored easily. Trace alignment can be used to explore the process in the early stages of analysis and to answer specific questions in later stages of analysis. Chapter 9 on *signature discovery* deals with discovering symptomatic patterns that can be correlated to different classes of cases in an event log.

One can look at the advancements in process mining proposed in this thesis from two perspectives as depicted in Figure 1.15. Chapters 6 and 7 take a process centric approach and deal with the discovery of process models and the enhancement of process models for additional insights. Chapters 8 and 9, on the other hand, take a case-centric approach and are concerned with uncovering insights by looking at event logs as raw traces. Figure 1.16 depicts the positioning of these chapters using the classification of Figure 1.3.

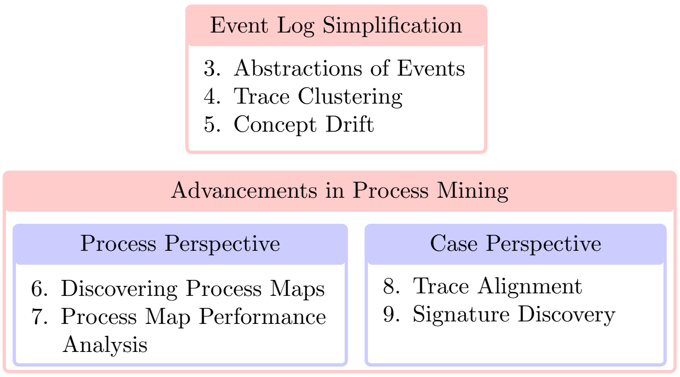


Figure 1.15: Core chapters of the thesis and their perspective of analysis.

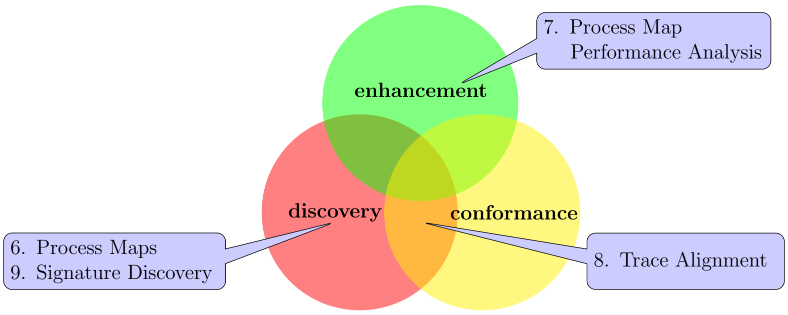


Figure 1.16: Positioning of chapters on advancements in process mining within the facets of process mining.

Part IV (Chapters 10–12) deals with applications and conclusions. It starts with Chapter 10 on *tool support* where we provide information on the ProM plug-ins that are developed within the context of this thesis. We present a few case studies substantiating the applicability of the various techniques proposed in this thesis in Chapter 11. Finally, Chapter 12 concludes the thesis and points out directions for future work.

Chapter 2
Preliminaries

In this chapter we introduce the necessary concepts and notations used in the remainder of this thesis.

2.1 Sets, Lists, Relations and Functions

Definition 2.1 (Set). A set is an unordered collection of unique objects, called *elements* of the set. An element of a set is sometimes called a *member* of the set. ⌟

A set can be represented by listing its elements between curly braces, e.g., $A = \{a_1, a_2, a_3, \ldots, a_n\}$. The order of the elements is irrelevant; thus, $\{a, b, c\} = \{c, a, b\}$. The symbol $\in$ is used to express that an element belongs to a set while its negation is represented by $\notin$, e.g., $a_2 \in A$ and $x \notin A$.

We use the following standard set representations

- $\mathbb{N}$ is the set of all natural numbers $\{1, 2, 3, \ldots, \}$
- $\mathbb{N}_0$ is the set of all natural numbers including zero
- $\mathbb{R}$ is the set of all real numbers. $\mathbb{R}^+$ and $\mathbb{R}^-$ denote the set of all positive and negative real numbers respectively. $\mathbb{R}_0^+$ is the set of all positive real numbers including zero, i.e., $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$.

An alternative way to define a set, called *set builder notation*, is by stating a property or a predicate that holds true only for the elements of the set, e.g., $A = \{x \in \mathbb{N} | x \text{ is divisible by } 5\}$ denotes the set of all multiples of 5. A set is called *finite* if it has finitely many elements. Otherwise, we say that the set is *infinite*.

Definition 2.2 (Null Set and Singleton). A set with no elements is called an empty set or a null set, and is represented by $\emptyset$ or $\{\}$. A set with only one element is called a *singleton*. ⌟

Definition 2.3 (Set Cardinality). If A is a finite set, then $|A|$ denotes the cardinality of A and is equal to the number of elements in A. For example, if $A = \{a, b, c\}$, then $|A| = 3$. ⌟

Definition 2.4 ((Proper) Subset and Equality). A set A is a subset of B, denoted by $A \subseteq B$, if and only if every element in A is also a member of set B. For example, if $A = \{a, b, c\}$ and $B = \{a, b, c, d, e\}$ then $A \subseteq B$.

Two sets, A and B, are equal, denoted by $A = B$, if and only if every element in A belongs to the set B and every element in B belongs to the set A, i.e., $A \subseteq B$ and $B \subseteq A$. For example, the sets $A = \{a, b, c, d\}$ and $B = \{c, a, d, b\}$ are equal.

A set A is a proper subset of B, denoted by $A \subset B$, if and only if $A \subseteq B$ and $A \neq B$. ⌟

Definition 2.5 (Union, Intersection, and Difference). The union of two sets A and B, denoted by $A \cup B$, is the set containing all elements in either A or B or both, i.e., $A \cup B = \{x | x \in A \text{ or } x \in B\}$. For example, if $A = \{a, b, c\}$ and $B = \{c, d, e\}$ then $A \cup B = \{a, b, c, d, e\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set containing all elements in both A and B, i.e., $A \cap B = \{x | x \in A \text{ and } x \in B\}$. For example, if $A = \{a, b, c\}$ and $B = \{c, d, e\}$ then $A \cap B = \{c\}$.

The difference between two sets A and B, denoted by $A \setminus B$, is the set containing all elements of A that are not elements of B. For example, if $A = \{a, b, c\}$ and $B = \{c, d, e\}$, then $A \setminus B = \{a, b\}$. ⌟

Definition 2.6 (Disjoint Sets). Two sets, A and B, are said to be *disjoint* or *mutually exclusive* if there are no common elements between them, i.e., if $A \cap B = \emptyset$. ⌟

Definition 2.7 (Powerset). The collection of all subsets of a set A is called the power set of A and is represented by 2^A. For example, if $A = \{a, b, c\}$, then $2^A = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, A\}$. If $|A| = n$ then $|2^A| = 2^n$. ⌟

Definition 2.8 (Partitioning). A partitioning of a set A is a collection $\mathcal{C}$ of non-overlapping non-empty subsets of A whose union is the set A. In other words, $\emptyset \notin \mathcal{C}$, $\bigcup_{P \in \mathcal{C}} P = A$ and for any $P_1, P_2 \in \mathcal{C}$, if $P_1 \neq P_2$, then $P_1 \cap P_2 = \emptyset$. For example, a partitioning of $A = \{a, b, c, d, e, f\}$ could be $\mathcal{C} = \{\{a, e\}, \{c\}, \{b, d, f\}\}$. ⌟

Definition 2.9 (Ordered Pair and Cartesian Product). The *Cartesian product* of two sets A and B, denoted by $A \times B$, is the set of all *ordered pairs* (a, b) such that $a \in A$ and $b \in B$. In other words, $A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$. For example, the Cartesian product of the sets $A = \{a, b\}$ and $B = \{1, 2, 3\}$ is $\{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$.

Two ordered pairs (a, b) and (b, a) are not equal unless $a = b$, i.e., $(a, b) = (a', b')$ if and only if $a = a'$ and $b = b'$.

The Cartesian product of n sets $A_1, A_2, \ldots, A_n$ defines the n-tuples over them.

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) | (a_1 \in A_1) \wedge (a_2 \in A_2) \wedge \cdots \wedge (a_n \in A_n)\}$$

⌟

Definition 2.10 (Relation). A binary *relation* R on a set A is a set of ordered pairs, usually defined by some sort of a rule, i.e., $R \subseteq A \times A$. For example, if $A = \{1, 2, 3\}$, then the 'less-than' relation on A is defined as $< = \{(1,2),(1,3),(2,3)\}$. We also write $a\ R\ b$ for $(a,b) \in R$ and $a \not R\ b$ for $(a,b) \notin R$. ⌟

Definition 2.11 ((Ir-)Reflexive, Symmetric, Transitive and Equivalence Relation). A binary relation R on a set A is said to be *reflexive* if every element is related to itself, i.e., $a\ R\ a$ for all $a \in A$. R is *irreflexive* or *anti-reflexive* if no element is related to itself. R is *symmetric* if for all a and b in A, if a is related to b, then b is related to a, i.e., $\forall a,b \in A, a\ R\ b \Rightarrow b\ R\ a$. R is said to be *transitive* if for any three elements $a,b,c \in A, a\ R\ b$ and $b\ R\ c \Rightarrow a\ R\ c$. R is an *equivalence* relation if and only if it is reflexive, symmetric, and transitive. ⌟

Definition 2.12 (Partial Order). A *partial order* on A is a irreflexive and transitive binary relation on A. In other words, a binary relation $\prec$ on a set A is called a partial order on A if for every $a \in A, a \nprec a$ and for every $a,b,c \in A$, if $a \prec b$ and $b \prec c$ then $a \prec c$. ⌟

Definition 2.13 (Function). A *(total) function* f from a set A into a set B, denoted $f : A \to B$, is a mapping that assigns to each element a of A an element $f(a) \in B$.

The set A is called the *domain* of the function $f : A \to B$ and is denoted by $Dom(f)$; the set B is called the *codomain* of f, and is denoted by $CoDom(f)$. The *range* of f is defined as $Range(f) = \{f(a) | a \in A\}$. If $A' \subseteq A$ and $f : A \to B$ is a function, then $f(A') = \{f(a) | a \in A'\}$. ⌟

Definition 2.14 (Function Equality). Two total functions are equal if they have the same domain, the same codomain, and if they take the same value on any element of the domain. In other words, two functions $f : A \to B$ and $g : X \to Y$ are equal if and only if $A = X$, $B = Y$, and $f(x) = g(x)$ for all $x \in A$. For example, the two functions $f(x) = \frac{x}{x^2}$ and $g(x) = \frac{1}{x}$ defined over $\mathbb{R} \setminus \{0\}$ are equal. ⌟

Definition 2.15 (Multiset or Bag). A *multiset* or a *bag* is an unordered collection of objects with repetitions allowed, e.g., $A = [a,a,a,b,b,c]$. The same multiset can also be represented as $[a^3, b^2, c^1]$ where the superscript signifies the multiplicity of an element. Given a set A, a multiset is defined as a cardinal valued function $M : A \to \mathbb{N}_0$ such that for each $x \in A, M(x)$ denotes the number of times x occurs. For a multiset $M : A \to \mathbb{N}_0$, we define $M(x) = 0$ if $x \notin A$. $\mathscr{B}(A) = A \to \mathbb{N}_0$ is the set of all multisets (bags) over A. ⌟

Definition 2.16 (Addition and Subtraction of Bags). Let $X : A_1 \to \mathbb{N}_0$ and $Y : A_2 \to \mathbb{N}_0$ be two bags. The sum of two bags, denoted by $S = X \uplus Y$, is defined as $S : A_1 \cup A_2 \to \mathbb{N}_0$ such that for all $a \in A_1 \cup A_2$, $S(a) = X(a) + Y(a)$. The difference of two bags, denoted by $S = X \setminus Y$, is defined as $S : A' \to \mathbb{N}_0$ where $A' = \{a \in A_1 | X(a) - Y(a) > 0\}$ and for all $a \in A', S(a) = X(a) - Y(a)$. The cardinality of a bag X, denoted by $|X|$, is defined as $|X| = \sum_{a \in A_1} X(a)$. ⌟

Definition 2.17 (Tuple (or List)). A tuple (or list) is an ordered collection of elements. An n-tuple is an ordered list of n elements. We represent a tuple by listing its elements between $\langle$ and $\rangle$, e.g., $\langle a,b \rangle, \langle a,a,c \rangle$. ⌟

Sequences

A sequence of length n, $\mathbf{s} = \langle s_1, s_2, \ldots, s_n \rangle \in A^*$, is an ordered list of symbols. Given a sequence $\mathbf{s}$,

- $|\mathbf{s}|$ denotes the length of the sequence $\mathbf{s}$
- $\mathbf{s}(i)$ represents the i^{th} symbol of $\mathbf{s}$
- $\mathbf{s}(i,j)$ represents the contiguous *subsequence* of $\mathbf{s}$ that starts at position i and ends at position j of $\mathbf{s}$ for $1 \leq i \leq j \leq |\mathbf{s}|$. If $i > j$, $\mathbf{s}(i,j)$ is the empty sequence $\langle\rangle$
- $\mathbf{s}^i$ represents the *prefix* of length i of $\mathbf{s}$, i.e., $\mathbf{s}^i = \mathbf{s}(1,i)$. The prefix is also referred to as the *head* of a sequence.
- $\mathbf{s}(i,|\mathbf{s}|)$ represents the *suffix* of $\mathbf{s}$ that begins at position i. The suffix is also referred to as the *tail* of a sequence.
- A *proper* prefix, suffix, or subsequence of $\mathbf{s}$ is respectively, a prefix, suffix, or subsequence that is neither the entire sequence $\mathbf{s}$ nor the empty sequence
- The *concatenation* of two sequences $\mathbf{s} = \langle s_1, s_2, \ldots, s_m \rangle$ and $\mathbf{t} = \langle t_1, t_2, \ldots, t_n \rangle$ is a new sequence $\mathbf{p}$, denoted by $\mathbf{p} = \mathbf{s} \diamond \mathbf{t}$, of length $m+n$ such that $\mathbf{p}(i) = \mathbf{s}(i)$ for $1 \leq i \leq m$ and $\mathbf{p}(i) = \mathbf{t}(i-m)$ for $m+1 \leq i \leq m+n$

Given a set A, A^* is the set of all finite sequences over A. A^n is the set of all finite sequences of length n over A. $A^n \subseteq A^*$. For convenience, we often represent a sequence $\mathbf{s} = \langle s_1, s_2, \ldots, s_n \rangle$ as a string $\mathbf{s} = s_1 s_2 \ldots s_n$.

2.2 Vectors and Matrices

Definition 2.18 (Matrix). A *matrix* A of dimension m by n (or $m \times n$) is a rectangular array of numbers written in the form

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \text{ or } A = [a_{ij}], \text{for } i = 1,2,\ldots,m \text{ and } j = 1,2,\ldots,n$$

The numbers a_{ij} are referred to as the *elements* of A. $A(i,j)$ denotes the number that is found in the i^{th} row and the j^{th} column. If $m = n$, the matrix is *square*. A $m \times 1$ matrix is a *column vector* and a $1 \times n$ matrix is a *row vector*. We use lower case arrowed letters to denote vectors, e.g., $\vec{v}$. ⌟

Definition 2.19 (Matrix Transposition). The *transpose* of an $m \times n$ matrix A, denoted by A^T, is an $n \times m$ matrix obtained by interchanging the rows and columns of A, i.e., $A(i,j) = A^T(j,i)$ for all $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$. ⌟

Definition 2.20 (Matrix Equality). Two matrices A and B are said to be equal, denoted by $A = B$, if they have the same dimension and their corresponding elements are equal, i.e., $a_{ij} = b_{ij}$ for all i and j. ⌟

Definition 2.21 (Scalar Multiplication and Division). The scalar multiplication of a matrix A and a real number α is defined to be a new matrix B, denoted by $B = \alpha A$ or $B = A\alpha$, whose elements b_{ij} are given by $b_{ij} = \alpha a_{ij}$. The division of a matrix A by a scalar $\alpha \neq 0$ is equivalent to multiplication by $1/\alpha$. ⌟

Definition 2.22 (Matrix Addition and Subtraction). The addition of two $m \times n$ matrices A and B is a new $m \times n$ matrix C, denoted by $C = A + B$, whose elements c_{ij} are given by $c_{ij} = a_{ij} + b_{ij}$, for all $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. Matrix subtraction is defined similarly by replacing + with −. If the dimension of the matrices A and B are different, then $A + B$ and $A - B$ are undefined. ⌟

Definition 2.23 (Matrix Product). The product of an $m \times n$ matrix A and an $n \times p$ matrix B is a new $m \times p$ matrix C, denoted by $C = AB$, whose elements c_{ij} are defined by $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. The product of two matrices A and B is not defined if the number of columns of A is not equal to the number of rows of B. ⌟

Definition 2.24 (Inner Product, Norm and Length). The *inner product* (or dot product or interior product) of two n-dimensional column vectors $\vec{u}$ and $\vec{v}$, denoted $\vec{u} \bullet \vec{v}$, is a scalar defined by

$$\vec{u} \bullet \vec{v} = \vec{u}^T \vec{v} = \vec{v}^T \vec{u} = \sum_{i=1}^{n} u_i v_i$$

The *Euclidean norm* or 2-norm of an n-dimensional vector $\vec{v}$ is a scalar obtained by taking the inner product of the vector with itself, i.e., $\|\vec{v}\| = \vec{v} \bullet \vec{v} = \sum_{i=1}^{n} v_i^2$

The *Euclidean length* or simply *length* of a vector $\vec{v}$ is the positive square root of its Euclidean norm, i.e., $+\sqrt{\|\vec{v}\|}$ ⌟

2.3 Graphs and Trees

Graphs play an important role in the representation, design, and analysis of process models. In this section, we introduce the basic concepts of graph theory.

Definition 2.25 (Graph). A directed graph $G = (V, E)$ consists of two finite sets V and $E \subseteq V \times V$. The elements of V are the *vertices* or *nodes* of G, and those of E, the *edges* or *links* of G. An edge $(u, v) \in E$ is directed from the source u to the destination v. The graph G can be considered to be *undirected* if the relation E is symmetric.

- Two vertices u and v of a graph G are said to be adjacent if there is an edge between u and v, i.e., $(u, v) \in E$.
- The adjacency matrix of a graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \dots, v_n\}$ is a binary $n \times n$ matrix $A = [a_{ij}]$ such that $a_{ij} = 1$ if $(v_i, v_j) \in E$ and $a_{ij} = 0$ otherwise.

⌟

Definition 2.26 (Walk, Path, and Cycle). Let $G = (V, E)$ be a graph. A sequence of vertices $\mathbf{w} = \langle v_1, v_2, \dots, v_k \rangle \in V^+$ of length $k > 0$ is a *directed walk* if $(v_{i-1}, v_i) \in E$

for $1 < i \leq k$. It is an *undirected walk* if either $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ for all $1 < i \leq k$.

A walk $\mathbf{w} = \langle v_1, v_2, \ldots, v_k \rangle$ is

- *closed*, if $(v_k, v_1) \in E$
- a *path*, if $v_i \neq v_j$ for all $i \neq j$ and $1 \leq i, j \leq k$
- a *cycle*, if it is closed, and $v_i \neq v_j$ for all $i \neq j$ and $1 \leq i, j \leq k$
- A graph is said to be *acyclic*, if it has no cycles

⌟

Definition 2.27 (Connected Graph). A graph $G = (V, E)$ is said to be *connected* if there exists a path from u to v in G, for every $u, v \in V$ without taking the direction of edges into account. G is said to be *strongly connected* if this property holds while respecting the direction of the edges. G is said to be *disconnected* if it is *not* connected.

⌟

Definition 2.28 (Subgraph). A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$, i.e., G' can be obtained from G by deleting some vertices (along with all its incident edges) and some edges. ⌟

Definition 2.29 (Tree). A *(rooted) directed tree* $T = (V, E)$, is a directed graph with a special vertex $r \in V$, designated as the *root* of the tree, such that for every $v \in V$, there is a directed path from r to v in T. We refer to a rooted directed tree as a directed tree henceforth. ⌟

Definition 2.30 (Parent, Child, Leaf, Ancestor, and Descendent). Given a directed tree $T = (V, E)$ and an edge $(u, v) \in E$, u is called the *parent* of v, and v is called the *child* of u. All nodes except the root have a unique parent. A *leaf* is a vertex with no children. For any path from u to v, u is an ancestor of v and v is a descendent of u. ⌟

Definition 2.31 (Height, Depth, and Degree). The *height of a node* in a tree is the length of the longest path from that node to a leaf. The *height of a tree* is the height of its root node. The *depth or level of a node* is the length of the path from the root to that node. The *degree of a node* is the number of children it has. ⌟

2.4 Suffix Tree

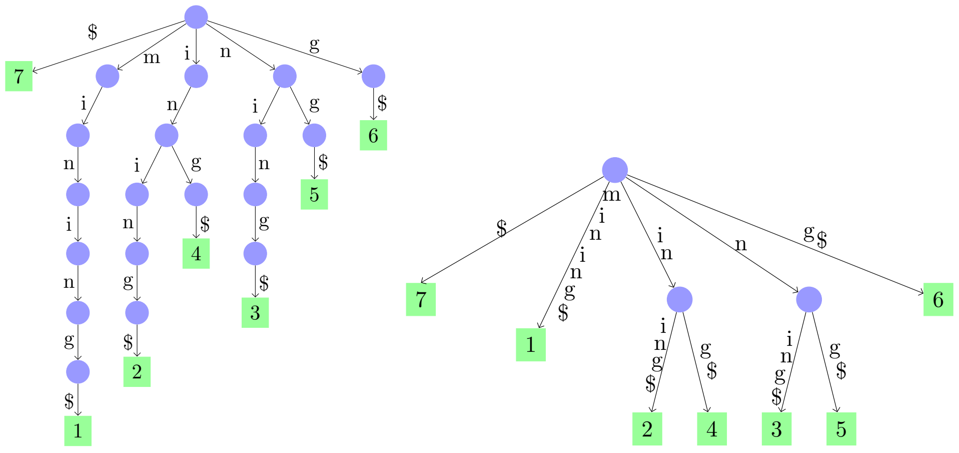
A *trie* is a rooted tree used for storing one or more sequences defined over an alphabet. Each edge in a trie is labeled by a symbol (in the alphabet) and every path from the root to a leaf corresponds to an input sequence. The concatenation of the edge labels from the root to a node is referred to as the *path label* of that node. A *suffix trie* is a trie storing all possible suffixes of a sequence $\mathbf{s}$, i.e., every path from the root to a leaf of a suffix trie corresponds to some $\mathbf{s}(i, |\mathbf{s}|)$ for $1 \leq i \leq |\mathbf{s}|$. A *suffix tree* is a compact suffix trie, where nodes with only one child are merged and the edge label replaced with the concatenation of labels of the merged edges. More formally, a suffix tree can be defined as

Definition 2.32 (Suffix Tree). A suffix tree for a sequence $\mathbf{s} \in \Sigma^+$ of length n is a rooted directed tree $ST = (V, E)$ with the following properties:

- ST has exactly n leaves labeled 1 to n
- Every internal node of ST is *branching*, i.e., has at least two children
- Every edge of ST is labeled with a non-empty subsequence $\mathbf{s}(i,j)$ for $1 \leq i \leq j \leq n$
- Edges leaving some node are labeled with subsequences starting with different symbols, i.e., for every node $u \in V$, there is only one $\mathtt{a}$-edge $u \overset{\mathtt{a}\diamond\alpha}{\rightarrow} v$ for some symbol $\mathtt{a} \in \Sigma$ and some subsequence $\mathbf{s}(i,j) = \alpha$.
- The concatenation of edge labels on the path from the root to a leaf labeled i $(1 \leq i \leq n)$ equals the suffix of the sequence $\mathbf{s}$ starting at i, i.e., $\mathbf{s}(i,n)$

⌟

Figure 2.1[1] depicts the suffix trie and the suffix tree for the sequence $\mathbf{s} = \langle \mathtt{m}, \mathtt{i}, \mathtt{n}, \mathtt{i}, \mathtt{n}, \mathtt{g}, \$ \rangle$. The leaf nodes are depicted as squares while the internal nodes are depicted as circles. The label inside a leaf node indicates the starting position of the suffix (corresponding to the path label of the leaf) in the sequence. For example, `ining$` is the suffix starting at position 2 in `mining$` while the one starting at position 6 is `g$`.



(a) Suffix Trie (b) Suffix Tree

Figure 2.1: The suffix trie and suffix tree for the sequence $\langle \mathtt{m}, \mathtt{i}, \mathtt{n}, \mathtt{i}, \mathtt{n}, \mathtt{g}, \$ \rangle$.

2.5 Event Logs

The starting point for process mining is the notion of an *event* and an *event log.* An event log captures the manifestation of events pertaining to the instances of a single process. A *process instance* is also referred to as a *case.* Each event in the log

[1]the sequence is represented as a string here

corresponds to a single case and can be related to an *activity* or a *task.* Events within a case need to be *ordered.* An event may also carry optional additional information like *time*, *transaction type*, *resource*, *costs*, etc. Timing information such as date and timestamp of when an event occurred is required to analyze the performance related aspects of the process. Resource information such as the person executing the activity is useful when analyzing the organizational perspective. We refer to these additional properties as *attributes.* To summarize,

- an event log captures the execution of a process.
- an event log contains process instances or cases.
- each case consists of an ordered list of events.
- each event can be associated exactly to a single case.
- events can have attributes such as activity, time, resource, etc.

We now formalize the various notions

Definition 2.33 (Event, Attribute, Classifier). Let $\mathcal{E}$ be the set of all event identifiers and AN be the set of all attributes. For an attribute $\mathtt{x} \in \text{AN}$, let $\mathcal{X}_{\mathtt{x}}$ denote its universe, i.e., the set of all possible values of $\mathtt{x}$. Given $\mathcal{E}$ and an attribute $\mathtt{x} \in \text{AN}$, $\#_{\mathtt{x}} : \mathcal{E} \to \mathcal{X}_{\mathtt{x}} \cup \{\perp\}$ denotes the value of attribute $\mathtt{x}$ for any event $e \in \mathcal{E}$. $\#_{\mathtt{x}}(e) = \perp$ for all attributes $\mathtt{x}$ not defined for e. ⌟

We use the standard attributes *activity, time, resource*, and *transaction type* for an event e. Let $\mathcal{A}$ be the set of activities, $\mathcal{T}$ be the time domain, $\mathcal{R}$ be the set of resources and TT be the set of transaction types: $\#_{activity}(e) \in \mathcal{A}$ signifies the activity associated with event e, $\#_{time}(e) \in \mathcal{T}$ indicates the timestamp of event e, $\#_{resource}(e) \in \mathcal{R}$ indicates the resource performing event e, and $\#_{trans}(e) \in \text{TT}$ indicates the transaction type associated with event e. The transaction type attribute refers to the life-cycle that an activity undergoes at various instances of time. A transaction life-cycle model that typically involves the state transitions *schedule, start, suspend, complete*, etc., is used.

Depending on the type of analysis, one can consider only one or a subset of attributes as a representation of an event. For example, in process model discovery, one can consider only the activity attribute or the activity attribute in conjunction with the transaction type as a representation for each event. As another example, one may consider only the resource attribute of an event for discovering organizational models. Thus, the same event log can be used for different types of analysis.

Definition 2.34 (Classifier). A *classifier* is a function that maps an event onto a representative *name* used for a particular type of analysis. For any event $e \in \mathcal{E}$, let $\underline{e}$ be the *name* of the event. For example, if events are identified by their activity names, then $\underline{e} = \#_{activity}(e)$. ⌟

Definition 2.35 (Case, Trace, Event log). Let $\mathscr{C}$ be the set of all case identifiers and AN be the set of all attributes (just like events, cases can also have attributes). For any case $c \in \mathscr{C}$ and attribute $\mathtt{x} \in \text{AN}$, $\widehat{\#}_{\mathtt{x}} : \mathscr{C} \to \mathcal{X}_{\mathtt{x}} \cup \{\perp\}$ indicates the value of attribute $\mathtt{x}$ for case c. A *trace* is a mandatory attribute of a case and represents a finite sequence of events $\mathbf{t} \in \mathcal{E}^*$ such that no two events in a trace are the same, i.e., for $1 \leq i < j \leq |\mathbf{t}|, \mathbf{t}(i) \neq \mathbf{t}(j)$. Furthermore, the events in a trace should follow a non-descending temporal order with respect to its time attribute if present, i.e., for $1 \leq i < j \leq |\mathbf{t}|, \#_{time}(\mathbf{t}(i)) \leq \#_{time}(\mathbf{t}(j))$.

For a case c, $\mathcal{E}_c$ denotes the set of events belonging to c, i.e., $\mathcal{E}_c = \widehat{\#}_{\texttt{trace}}(c)$. An *event log* is a set of cases $\mathcal{L} \subseteq \mathscr{C}$ such that each event occurs only once, i.e., for any two cases $c_1, c_2 \in \mathscr{C}, c_1 \neq c_2 : \mathcal{E}_{c_1} \cap \mathcal{E}_{c_2} = \varnothing$.

If the activity name is used as a classifier, then a trace corresponds to a sequence of activities. In such a scenario, two or more cases can correspond to the same activity sequence. Therefore, an event log is a multi-set of traces. ⌟

Definition 2.36 (Simple Event log). Let $\mathcal{A}$ be the set of activities. A *simple* trace $\mathbf{t}$ is a sequence of activities, i.e., $\mathbf{t} \in \mathcal{A}^*$. A *simple* event log $\mathcal{L}$ is a multi-set of traces over $\mathcal{A}$, i.e., $\mathcal{L} \in \mathscr{B}(\mathcal{A}^*)$. ⌟

One can convert an event log specified as per Definition 2.35 to a simple event log using classifiers. Figure 2.2 depicts an example of an event log. All cases in an event log $\mathcal{L}$ can be converted into sequences of activity names by using the classifier, $\#_{activity}(e)$. If we apply this classifier to the cases in Figure 2.2, then we obtain the simple event log:

$$\begin{aligned}\mathcal{L} = [&\langle \textit{register}, \textit{high insurance check}, \textit{high medical history check}, \textit{contact hospital},\\ &\quad \textit{decide}, \textit{prepare notification}, \textit{send notification by email}, \textit{archive}\rangle,\\ &\langle \textit{register}, \textit{low insurance check}, \textit{low medical history check}, \textit{decide}, \textit{prepare-}\\ &\quad \textit{notification}, \textit{send notification by post}, \textit{archive}\rangle,\\ &\langle \textit{register}, \textit{high insurance check}, \textit{send questionnaire}, \textit{high medical history check},\\ &\quad \textit{receive response}, \textit{contact hospital}, \textit{decide}, \textit{prepare notification}, \textit{send notification-}\\ &\quad \textit{by email}, \textit{archive}\rangle,\\ &\langle \textit{register}, \textit{send questionnaire}, \textit{low insurance check}, \textit{low medical history check},\\ &\quad \textit{decide}, \textit{prepare notification}, \textit{send notification by post}, \textit{skip response}, \textit{archive}\rangle,\\ &\quad \ldots]\end{aligned}$$

Instead, if we had chosen the resource classifier $\#_{resource}(e)$, the simple event log turns out to be:

$$\begin{aligned}\mathcal{L} = [&\langle \textit{Bob}, \textit{Alice}, \textit{Alice}, \textit{Alice}, \textit{Wil}, \textit{Bob}, \textit{Bob}, \textit{Bob}\rangle,\\ &\langle \textit{Anita}, \textit{Anu}, \textit{Anu}, \textit{JC}, \textit{Anita}, \textit{Anita}, \textit{Anita}\rangle,\\ &\langle \textit{Mike}, \textit{Kate}, \textit{Mike}, \textit{Kate}, \textit{Sara}, \textit{Peter}, \textit{Chase}, \textit{Sasha}, \textit{Sasha}, \textit{Mike}\rangle,\\ &\langle \textit{Tom}, \textit{Harry}, \textit{Mika}, \textit{Mika}, \textit{Mark}, \textit{Dash}, \textit{Kim}, \textit{Tom}, \textit{Kate}\rangle,\\ &\quad \ldots]\end{aligned}$$

Projection using the resource classifier can be used in mining organizational models, social networks, etc.

case id	event id	properties				
		time stamp	activity	resource	cost	...
1	10001001	01-10-2010:09:32	register	Bob	10	...
	10001002	02-10-2010:11:17	high insurance check	Alice	15	...
	10001003	02-10-2010:16:43	high medical history check	Alice	15	...
	10001004	03-10-2010:13:54	contact hospital	Alice	20	...
	10001005	04-10-2010:18:32	decide	Wil	150	...
	10001006	05-10-2010:09:05	prepare notification	Bob	10	...
	10001007	05-10-2010:10:13	send notification by email	Bob	10	...
	10001008	05-10-2010:10:44	archive	Bob	10	...
2	10002001	01-10-2010:11:01	register	Anita	10	...
	10002002	04-10-2010:14:23	low insurance check	Anu	12	...
	10002003	05-10-2010:10:37	low medical history check	Anu	12	...
	10002004	08-10-2010:08:16	decide	JC	50	...
	10002005	10-10-2010:14:05	prepare notification	Anita	10	...
	10002006	11-10-2010:15:13	send notification by post	Anita	10	...
	10002007	14-10-2010:09:41	archive	Anita	10	...
3	10001231	11-11-2010:10:32	register	Mike	11	...
	10001232	12-11-2010:12:13	high insurance check	Kate	17	...
	10001233	12-11-2010:13:14	send questionnaire	Mike	23	...
	10001234	22-11-2010:14:23	high medical history check	Kate	17	...
	10001235	22-11-2010:15:41	receive response	Sara	17	...
	10001236	03-12-2010:06:54	contact hospital	Peter	23	...
	10001237	04-12-2010:08:12	decide	Chase	100	...
	10001238	15-12-2010:13:05	prepare notification	Sasha	10	...
	10001239	15-12-2010:17:13	send notification by email	Sasha	10	...
	10001240	17-12-2010:18:14	archive	Mike	10	...
4	10006711	03-01-2011:13:01	register	Tom	10	...
	10006712	04-01-2011:11:33	send questionnaire	Harry	12	...
	10006713	06-01-2011:09:43	low insurance check	Mika	12	...
	10006714	15-01-2011:11:17	low medical history check	Mika	12	...
	10006715	18-01-2011:19:16	decide	Mark	80	...
	10006716	20-01-2011:17:05	prepare notification	Dash	10	...
	10006717	31-01-2011:05:13	send notification by post	Kim	10	...
	10006718	04-02-2011:17:23	skip response	Tom	12	...
	10006719	14-02-2011:09:41	archive	Kate	10	...
...	...	...	...	...	...	...

Figure 2.2: An example event log.

In the remainder, we will use whatever notion is appropriate. Definition 2.35 specifies a generic description of an event log that can be used for various purposes, e.g., mining organizational models, performance analysis, etc. Definition 2.36 on the other hand specifies a simple description of an event log without any attributes. This format is adequate for explaining most of the concepts in this thesis (e.g., abstractions of events, trace clustering, trace alignment, etc.) that are not using the information stored in additional attributes. As mentioned earlier, one can convert any event log $\mathcal{L}$ into a simple event log. Simple event logs focus on a single attribute, with the attribute typically being the activity name. However, the simple format does not allow us to point at a specific event.

2.6 Representing and Storing Event Logs

Event logs from different systems and organizations can be stored in different formats, e.g., databases, plain text files, etc. For process mining applications, a common event log format based on a process meta model, called the MXML format [244], has been proposed. This has been followed by a more recent advancement, called the XES [92]. XES is adopted as the standard by the IEEE Task Force on Process Mining. In this section, we give a brief description of these two formats for storing and representing event logs.

2.6.1 The MXML Format

In order to store event logs, the MXML (**M**ining**XML**) format has been defined [244]. Figure 2.3 depicts the schema of the MXML format as an UML 2.0 class diagram. An event log corresponds to the element `WorkflowLog` and can contain event data pertaining to one or more processes (element `Process`). A workflow log can optionally have a `Source` element, which describes the system from which the log has been imported, e.g., X-ray machine Y in hospital Z at Eindhoven, Staffware system, etc. Each `Process` in a workflow log can contain an arbitrary number of `ProcessInstances`. Each `ProcessInstance` captures an ordered sequence of events pertaining to that case. Events are recorded as `AuditTrailEntries` and should contain two mandatory elements, viz., `WorkflowModelElement` and `EventType`. `WorkflowModelElement` corresponds to the activity or task name to which this event corresponds to and `EventType` corresponds to the transaction type of the activity. An `AuditTrailEntry` may optionally also contain additional information such as the `Timestamp` indicating the date and time when the event actually happened and `Originator` indicating the resource that performed this task. All elements (`WorkflowLog`, `Source`, `Process`, `ProcessInstance`, and `AuditTrailEntry`) can in addition have an optional data element, which basically groups all attributes in the form of *key-value* pairs of strings.

Figure 2.4 depicts a fragment of a log pertaining to an X-ray machine in MXML format. The events here correspond to the *commands* that were executed on the machine. We can see attributes such as `Group`, `MainMenu` and `Procedure` defined under the `Data` element for `AuditTrailEntries` (i.e., events). These attributes define abstractions over the commands at different levels of granularity and satisfy the following ordering relation `Command` ⪯ `Group` ⪯ `Procedure` ⪯ `MainMenu`. Domain knowledge such as this as well as additional information relevant to the event can be stored as data elements. It is important to note that these attributes are optional and that different events can have different subset of these attributes, e.g., the audit trail entries corresponding to the `WorkflowModelElement` 'Login' has two data attributes while the one corresponding to 'Select Procedure' has no data element.

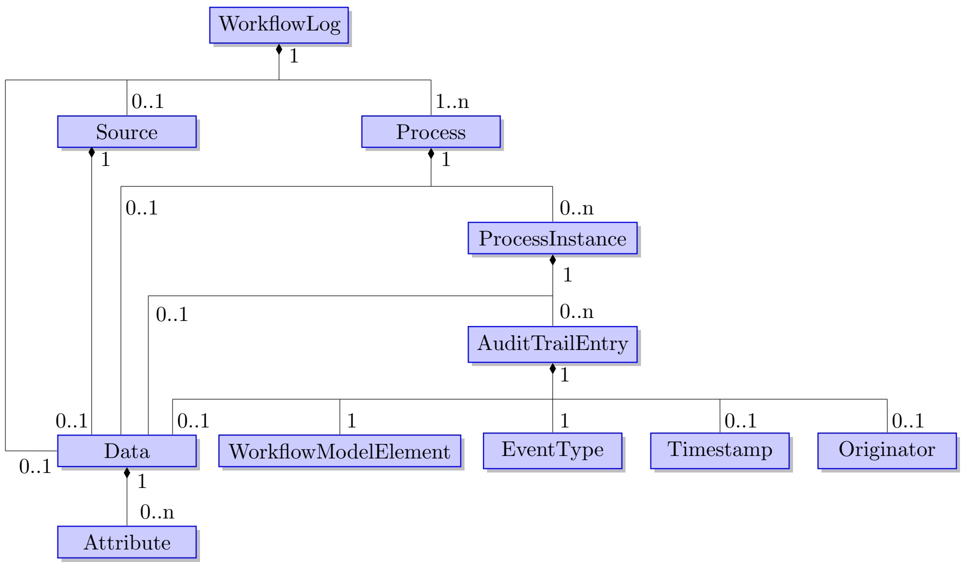


Figure 2.3: Schema of the MXML format as an UML 2.0 class diagram [93].

Limitations of MXML: While MXML tries to capture the basic notions of an event log, it suffers from a few limitations as highlighted in [91].

- The *nomenclature* used as elements and attributes has an inherent inclination towards capturing an event log emanating from well-structured processes and workflow like environments. The basic entities of event logs generated from flexible environments and non-workflow like environments such as high-tech system logs and product usage may not have a direct mapping to the elements of MXML. For example, when analyzing the system failures, the error/warning events of an X-ray machine event log need to be accommodated as a `WorkflowModelElement` (which usually captures the notion of a task or an activity). Such a non-intuitive mapping can lead to misinterpretations and compromises on the comprehensibility.
- MXML has *no concept for hierarchies and taxonomies* thus making it hard to translate logs with hierarchies and expressing domain knowledge, e.g., incorporating the various levels of abstraction over commands in an X-ray machine.

  The Semantically Annotated MXML format (SA-MXML) [55] is an extension of the MXML format that allows for the provision of references between elements in the log and domain concepts, in the form of ontologies, for the logged information. This enables the analysis of the log at different levels of abstraction albeit at an additional overhead in the analysis. However, defining ontologies is not a trivial task.
- MXML has *no straightforward extensibility* to incorporate additional knowledge. Additional information is often encoded as data elements in the form of key-

```
<WorkflowLog ...>
  <Source program="X-ray machine"/>
  <Process id="FieldService\FSCommands">
    <ProcessInstance id="3439242_2007-04-03_2007-04-03_1">
      <AuditTrailEntry>
        <Data>
          <Attribute name="Main Menu">Field Service Window</Attribute>
          <Attribute name="Procedure">Service Login</Attribute>
        </Data>
        <WorkflowModelElement>Login</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2007-04-03T15:48:09.000+01:00</Timestamp>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <Data>
          <Attribute name="Group">SelectFluoFlavour</Attribute>
          <Attribute name="Main Menu">Acquisition Presetting</Attribute>
        </Data>
        <WorkflowModelElement>SelectFluoFlavour</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2007-04-03T15:48:13.000+01:00</Timestamp>
      </AuditTrailEntry>
            :
            :
      <AuditTrailEntry>
        <WorkflowModelElement>SelectProcedure</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2007-04-03T16:28:14.000+01:00</Timestamp>
      </AuditTrailEntry>
            :
            :
      <AuditTrailEntry>
        <Data>
          <Attribute name="Group">ClinicalUI</Attribute>
          <Attribute name="Main Menu">Field Service Window</Attribute>
          <Attribute name="Procedure">Exit</Attribute>
        </Data>
        <WorkflowModelElement>LogOff</WorkflowModelElement>
        <EventType>start</EventType>
        <Timestamp>2007-04-03T17:52:40.000+01:00</Timestamp>
      </AuditTrailEntry>
    </ProcessInstance>
            :
  </Process>
</WorkflowLog>
```

Figure 2.4: A log fragment in MXML format.

value attributes. However, this additional information is less usable and is often lost due to the lack of any specific semantics associated to these attributes.

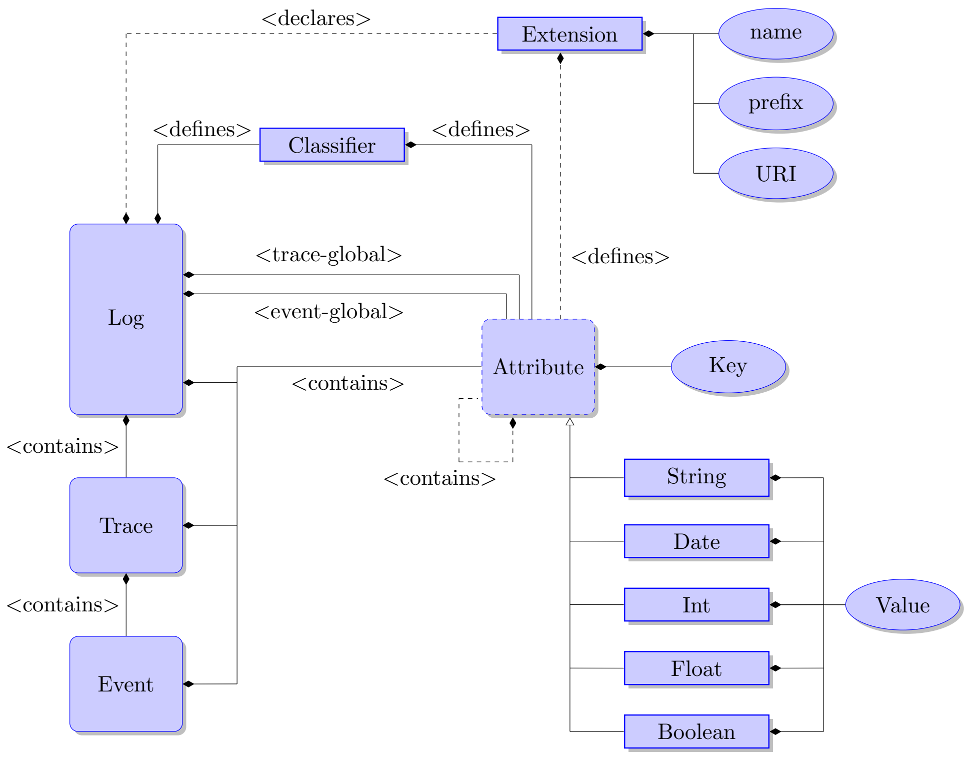


Figure 2.5: Meta model of XES [92].

2.6.2 The XES Standard

To overcome the shortcomings mentioned above, a new event log standard, called the eXtensible Event Stream (XES) [92], that is less restrictive and highly extensible has been proposed. Figure 2.5 depicts the XES meta model expressed as an UML class diagram. An XES log contains any number of traces from a single process[2]. Each trace captures the sequential list of events corresponding to a particular case. Attributes can be defined for a log, trace, and event and can be nested, i.e., an attribute can contain other attributes. XES supports five core data types, viz., *String, Date, Int, Float*, and *Boolean* corresponding to standard XML built-in data types `xs:string`, `xs:dateTime`, `xs:int`, `xs:float`, and `xs:boolean` respectively. Attributes that are mandatory need to be declared as *global* attributes.

Semantics for attributes are specified through *extensions*. XES defines five standard extensions:

- *Concept:* The *concept extension* is defined for traces and events and captures their *name*. For traces, it typically represents the case identifier while for events it represents the activity name, i.e., $\#_{activity}(e)$ for all $e \in \mathcal{E}$. This extension

[2]MXML allows multiple processes to be specified in a single file while XES restricts this to only one process.

can also define an *instance* attribute to distinguish between different instances of an activity.

- *Time:* The *time extension* is defined for events and captures their timestamp of type `xs:dateTime` (both the date and time need to be specified). This corresponds to $\#_{time}(e)$ for all $e \in \mathcal{E}$.
- *Organization:* The *organization extension* is defined for events and captures the organizational perspective of a process. Three attributes, viz., *resource, role, and group* are defined for this extension. Role characterizes the capabilities of resources while group classifies the position of resources in an organization.
- *Lifecycle:* The *lifecycle extension* is defined for events and signifies their transaction type, i.e., $\#_{trans}(e)$ for all $e \in \mathcal{E}$.
- *Semantic:* The *semantic extension* caters to the enhancements proposed in SA-MXML and defines the *modelReference* attributes for all the elements in the log.

Furthermore, XES supports the concept of *classifiers* defined earlier. The reader is referred to [92] for a detailed description of the XES standard. Figure 2.6 depicts a log fragment in XES format.

```
<log xes.version="1.0" xes.features="nested-attributes">
   <extension name="Concept" prefix="concept" uri="http://.../concept.xesext"/>
   <extension name="Semantic" prefix="semantic" uri="http://.../semantic.xesext"/>
   <extension name="Time" prefix="time" uri="http://.../time.xesext"/>
   <extension name="Organizational" prefix="org" uri="http://.../org.xesext"/>
   <extension name="Lifecycle" prefix="lifecycle" uri="http://.../lifecycle.xesext"/>
   <global scope="trace">
       <string key="concept:name" value="name"/>
   </global>
   <global scope="event">
       <string key="concept:name" value="name"/>
       <string key="time:timestamp" value="2007-04-03T16:00:00.000+00:00"/>
   </global>
   <classifier name="Activity" keys="concept:name"/>
   <trace>
       <string key="concept:name" value="3439242_207-04-03_2007-04-03_1"/>
       <event>
           <string key="concept:name" value="Login"/>
           <string key="lifecycle:transition" value="complete"/>
           <date key="time:timestamp" value="2007-04-03T15:48:09.000+01:00"/>
           <string key="Group" value="Service Login"/>
           <string key="Main Menu" value="Field Service Window"/>
           <string key="Procedure" value="Service Login"/>
       </event>
       <event>
           <string key="concept:name" value="SelectFluoFlavour"/>
           <string key="lifecycle:transition" value="complete"/>
           <date key="time:timestamp" value="2007-04-03T15:48:13.000+01:00"/>
           <string key="Group" value="SelectFluoFlavour"/>
           <string key="Main Menu" value="Acquisition Presetting"/>
       </event>
           :
       <event>
           <string key="concept:name" value="SelectProcedure"/>
           <string key="lifecycle:transition" value="complete"/>
           <date key="time:timestamp" value="2007-04-03T16:28:14.000+01:00"/>
       </event>
           :
       <event>
           <string key="concept:name" value="LogOff"/>
           <string key="lifecycle:transition" value="complete"/>
           <date key="time:timestamp" value="2007-04-03T17:52:40.000+01:00"/>
           <string key="Group" value="ClinicalUI"/>
           <string key="Main Menu" value="Field Service Window"/>
           <string key="Procedure" value="Exit"/>
       </event>
   </trace>
      :
      :
</log>
```

Figure 2.6: A log fragment in XES format.

2.7 Process Modeling Formalisms

Processes can be represented (modeled) using many different techniques/languages, e.g., BPMN (Business Process Modeling Notation), UML activity diagrams, EPCs (Event-driven Process Chains), Petri nets, etc. These techniques/languages differ in their expressive power and formal semantics. In this section, we briefly present four different process modeling notations that are used in this thesis. Note that we only provide an informal introduction for these modeling languages; however, we provide pointers for their formal definitions. The workflow modeling language Yet Another Workflow Language (YAWL) is used to represent the running examples used in this thesis (cf. Chapters 3 and 5). Colored Petri nets (CPNs) are also used to model these running examples. Moreover, the CPNs are used to generate event logs using simulation for our experiments. Fuzzy models are used when discovering process maps (cf. Chapter 6). Fuzzy models are also extended with performance annotations (cf. Chapter 7).

2.7.1 Petri Nets and Workflow Nets

Petri nets [172], named after Carl Adam Petri, are one of the commonly used process modeling formalisms. A (classical) Petri net is a directed bipartite graph with two types of nodes, *places* and *transitions*. Places are depicted by *circles* while transitions are depicted by *rectangles*. A place may contain zero or more tokens, depicted as black dots. Places and transitions in a Petri net are connected by directed arcs with a constraint that connections between nodes of the same type are not permitted. In other words, places can be connected to transitions and transitions can be connected to places, but places cannot be connected to places and transitions cannot be connected to transitions. A place p is called an *input place* of a transition t if and only if there is a directed arc from p to t. If a directed arc exists from t to p then p is called an *output place* of the transition t. Petri nets are much more than simple graphs. They have execution semantics. The distribution of tokens over places indicates the *state* of the Petri net. A transition t is said to be *enabled* if each of its input places contains a token. Transitions that are enabled may *fire*. If a transition t fires, then a token is consumed from each of its input places and a token is produced in each of its output places. Thus the firing of transitions changes the state of the net. The behavior of a Petri net can be analyzed for interesting properties, e.g., soundness, deadlocks, etc. The interested reader is referred to [157] for an extensive review on Petri nets.

The modeling capability combined with analyzability make Petri nets a powerful formalism for modeling business processes. Workflow nets (WF-nets) are a subclass of Petri nets that are typically used to model business processes. A WF-net is a Petri net that has (a) a single *start* place, (b) a single *end* place, and (c) every node is on some path from start to end. In the context of business processes, places signify *conditions* and transitions signify *tasks* or *activities*. Figure 2.7 depicts an example WF-net model of an insurance claim process. The place p_0 is the start place and the place p_{13} is the end place. Initially, there is one token in the place p_0, indicating

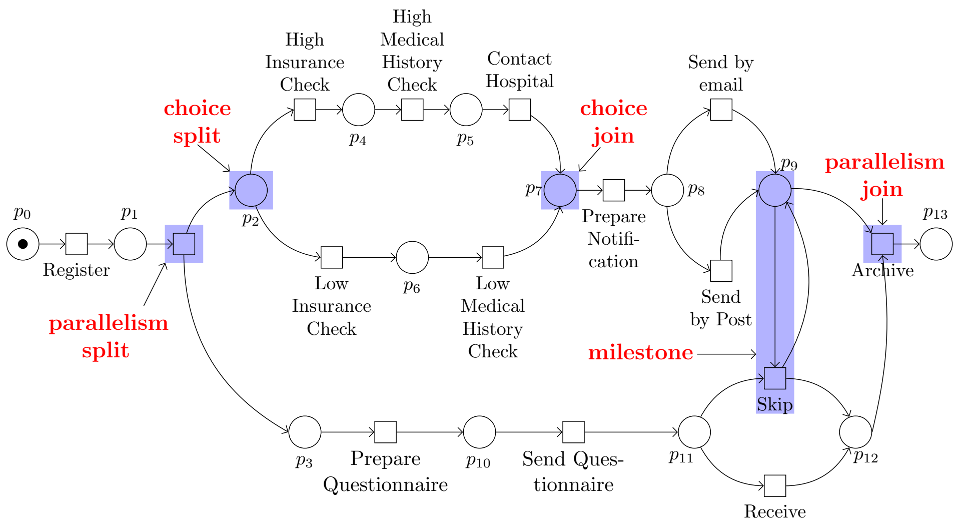


Figure 2.7: Petri net modeling of an insurance claim process.

that there is a case to be processed. Once the transition with the label `Register` fires, the token from the left most place will be consumed, and another token will be produced and put in the output place of transition `Register`, i.e., p_1. The next transition, highlighted in the figure as *parallelism split*, activates multiple branches at the same time. As soon as this transition fires, two tokens are produced (one for each of the output places p_2 and p_3), which enables the independent (concurrent) action flow in these two branches. Once a token is in place p_2, there is a *race* between the transitions `Low Insurance Check` and `High Insurance Check`, because both the transitions are enabled but only one of them can consume this token. *Choice* constructs are modeled in this fashion in Petri nets. The response for the questionnaire can be skipped only after the customer is notified about the status (i.e., when there is a token in p_9). This corresponds to the *milestone* pattern. A case is considered to be completed if a token reaches the place p_{13}. The interested reader is referred to [223] for a rigorous introduction on the modeling of business processes using Petri nets.

2.7.2 Yet Another Workflow Language (YAWL)

It was observed that most existing workflow languages had a poor support for the workflow patterns [228]. Workflow patterns refer to the essential and recurring forms/structures addressing business requirements. Workflow patterns can be divided into four perspectives:

- *control flow*: The control-flow patterns [195] capture aspects related to control-flow dependencies between various tasks, e.g., sequence, parallelism (AND), choice (such as XOR and deferred choice), etc.

- *data*: The data patterns [193] deal with data-related issues such as the scope (e.g., task data, case data, etc.), interaction (e.g., task-to-task, case-to-case, etc.), and transfer (e.g., transfer by value, transfer by reference, etc.). In addition, data-routing patterns exist that deal with the influence of data on routing the process execution (e.g., pre-condition, post-condition, etc.).

- *resource*: The resource patterns [194] capture the various ways in which resources are represented and utilized in workflows, e.g., capability-based distribution, random allocation, etc.

- *exception handling*: The exception handling patterns [196] deal with the various causes of exceptions and the various actions that ought to be taken as a result of exceptions, e.g., deadline expiry, resource unavailability, etc.

YAWL [224] is an *executable* workflow language that has been specifically designed with an objective to support most of the workflow patterns using a language as simple as possible. The genesis of YAWL lies with Petri nets but it is more expressive than Petri nets with several constructs added to the formalism to extend for the pattern support. Some of the notable extensions are the *OR-join* [269], *cancelation set*, and *multi-instance* tasks (execute the task multiple times). In addition, YAWL provides some syntactical elements to intuitively capture workflow patterns, e.g., simple choice/merge is represented with an XOR split/join.

Figure 2.8 depicts the YAWL model of the insurance claim process. We can see that the YAWL model is compact in its representation when compared to the Petri net model. This is due to the various syntactic elements defined in YAWL. Unlike Petri nets, YAWL allows two tasks to be connected directly. Most places have thus disappeared. Furthermore, YAWL allows for the specification of perspectives other than the control-flow perspective, viz., data and resource perspectives. For example, we can specify resources or roles (of resources) that can execute a task. Tasks that need to be performed by human resources can be represented with the 'manual icon' as indicated in Figure 2.8.

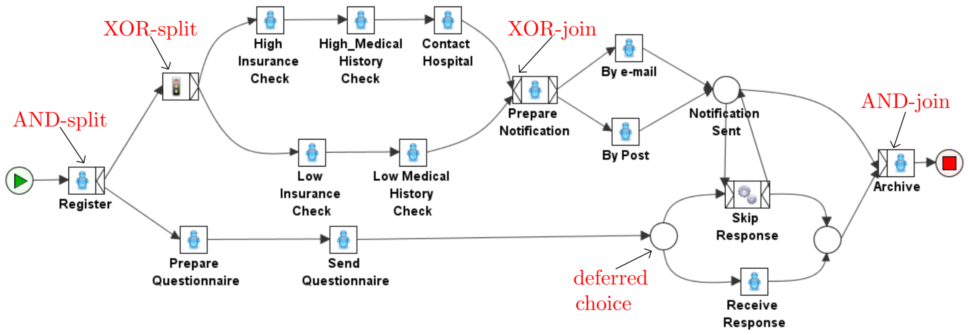


Figure 2.8: YAWL model of the insurance claim process.

2.7.3 Colored Petri Nets (CPNs)

Colored Petri Nets [117] are a modeling formalism that extend Petri nets with data, time, and hierarchy. CPNs extend classical nets with the power of programming languages. For example, CPN Tools[3] [117, 118, 180] uses the functional programming language Standard ML (SML), to describe place types, guards, and arc inscriptions. There are three primary differences between classical Petri nets and CPNs:

- Unlike Petri nets where the tokens in a place are indistinguishable, each token in a CPN is associated with a data type and value (the data value is called the *color* of the token), which makes them distinguishable. The assignment of a value to a token is called a *binding.* The number of tokens and the token colors on the individual places together represent the *state* of the system, which is also called as the *marking* of the CPN model.
- Arcs can have *expressions*, which are written in SML and built from variables, constants, operators, and functions. All arc expressions should evaluate to a single token color or a multiset of token colors. The arc expressions on the input arcs of a transition determine whether the transition is enabled, i.e., is able to occur in a given marking.
- Transitions can have a *guard*, which is a boolean expression. Guards put extra constraints on the enabling of bindings for the transition. The guard must evaluate to true for the binding to be enabled, otherwise the binding is disabled and cannot occur.

The hierarchy extension supported by CPN is extremely handy when dealing with large and/or complex models. Models can be decomposed into smaller manageable modules that can be reused. The time extension allows for the analysis of performance aspects via simulation. Figure 2.9 depicts an example of a system modeled as a colored Petri net.

CPN Tools is a tool for editing, simulating, and analyzing colored Petri nets. Two modes of simulation are supported in CPN tools (i) *interactive mode* and (ii) *automated mode.* In the interactive mode, the simulator calculates the set of all *enabled* transitions in each marking encountered. The user can then choose an enabled transitions and one of its bindings and thus fire transitions step-by-step. In the automated mode, which is analogous to program execution, the simulator performs all the calculations and makes all the choices randomly. The automated mode also allows the user to specify breakpoints (as the number of steps of execution). The simulator stops once the breakpoint is reached and the user can inspect the marking reached by the model or look at the simulation report.

2.7.4 Fuzzy Models

The motivation for Fuzzy models [91] emanated from the need for representing highly unstructured and flexible processes. The goal is to have models that are comprehensible rather than being precise. Precise representations of flexible and unstructured

[3]http://cpntools.org

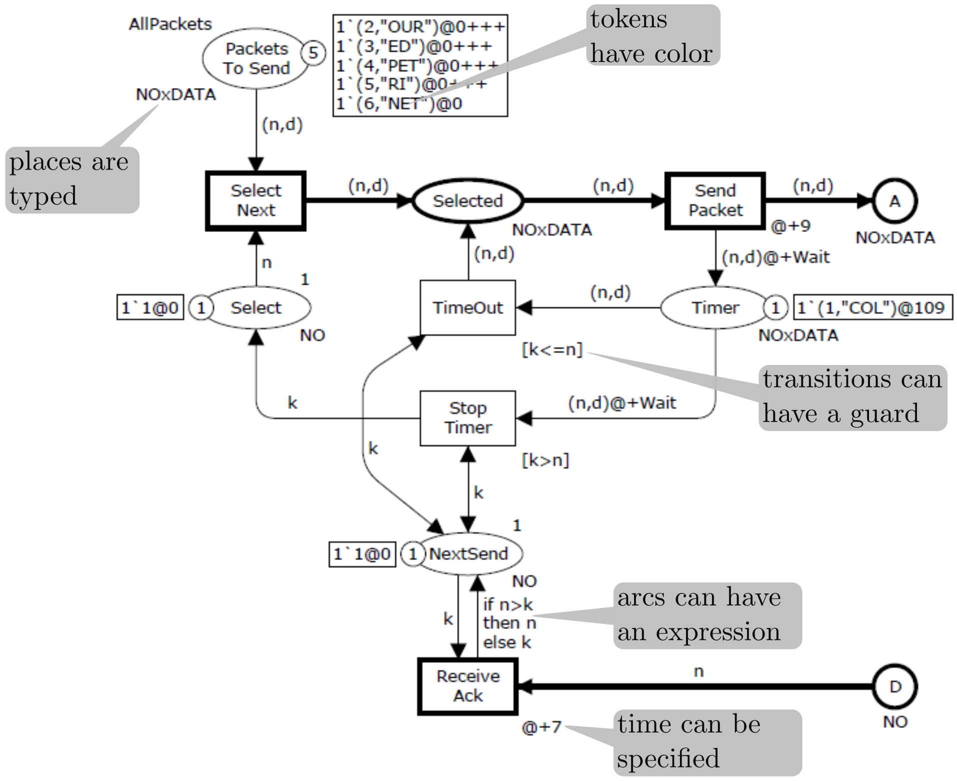


Figure 2.9: Example of a CPN model (adapted from [117]).

processes often tend to be *spaghetti-like.* Fuzzy models, designed using the *map metaphor*, borrow universal concepts such as *aggregation*, *abstraction*, *emphasis*, and *customization* from cartography. In Fuzzy models, coherent clusters of low-level information are grouped together as an *aggregate* node while insignificant information is *abstracted* (omitted) and significant information is *emphasized* (highlighted by thickness, color, etc.) [94].

Fuzzy models do not distinguish between AND-split (parallelism), XOR-split (simple choice), and OR-split (multi-choice). If a task (represented as a node) in a Fuzzy model has multiple successor tasks, then all of these successors will be *activated* once the task has been executed. However, they *need not* be executed. The relaxed execution semantics [91] for Fuzzy models are defined as follows:

- *Initialization:* The process, captured as a Fuzzy model, can start at any node in the model. Different cases may start at different nodes.
- *Termination:* The process is considered to be terminated whenever no further nodes are executed, i.e., there are no explicit ending point(s) and remaining enabled nodes (if any) are ignored.

- *Branch Semantics:* Every node in a Fuzzy model has an AND-split semantics, i.e., upon execution of a node, all its successors are enabled.
- *Join Semantics:* Every node in a Fuzzy model has memory-less XOR-join semantics, i.e., it can be executed as soon as it has been enabled by any of its predecessor nodes (but it does not "remember" how often it has been enabled).

The reader is referred to [91] for a detailed description of the semantics and evaluation of Fuzzy models with respect to the workflow patterns. Figure 2.10 depicts the Fuzzy model (mined using the Fuzzy miner plug-in in ProM) of the insurance claim process. The YAWL model of the process in Figure 2.8 specifies that the prepare notification task is an XOR-split (simple choice). This implies that only one of the two tasks, viz., By e-mail or By Post, would be enabled. However, according to the relaxed Fuzzy semantics both tasks are enabled (see Figure 2.10).

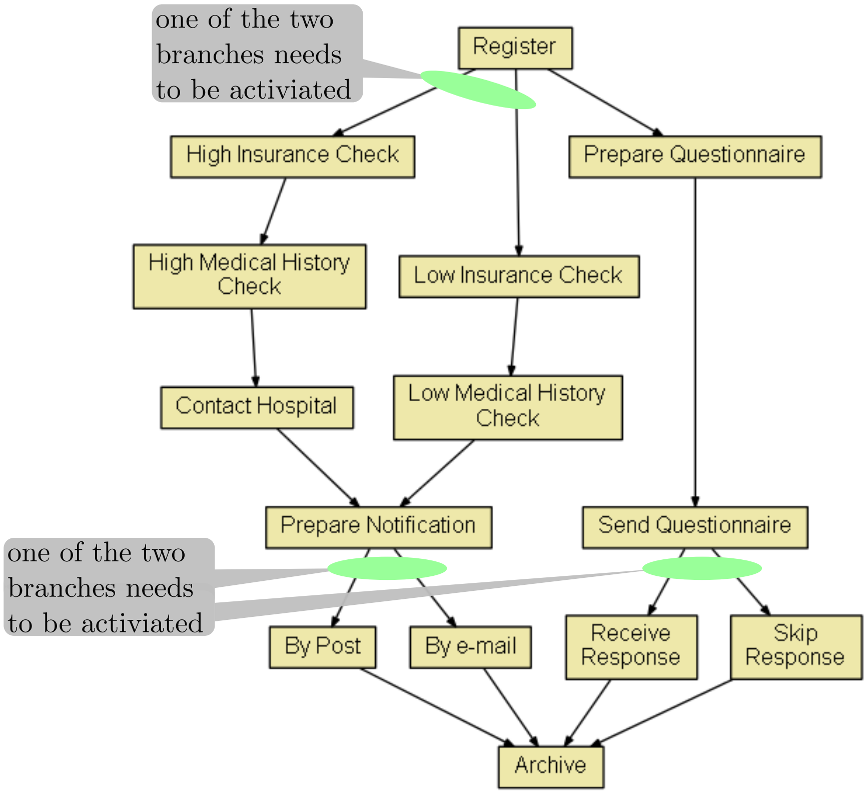
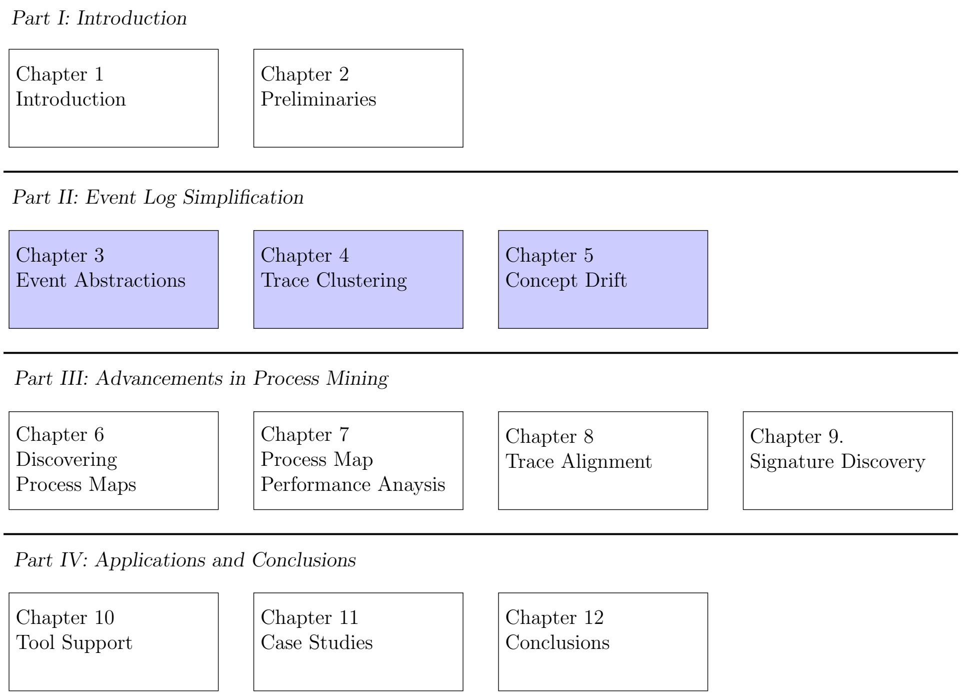


Figure 2.10: Fuzzy model of the insurance claim process.

Part II
Event Log Simplification


Part I: Introduction
Chapter 1 Introduction
Chapter 2 Preliminaries
Part II: Event Log Simplification
Chapter 3 Event Abstractions
Chapter 4 Trace Clustering
Chapter 5 Concept Drift
Part III: Advancements in Process Mining
Chapter 6 Discovering Process Maps
Chapter 7 Process Map Performance Anaysis
Chapter 8 Trace Alignment
Chapter 9. Signature Discovery
Part IV: Applications and Conclusions
Chapter 10 Tool Support
Chapter 11 Case Studies
Chapter 12 Conclusions

After providing the motivation for the work done in this thesis and the preliminaries needed for a good understanding of the concepts (to be) presented in this thesis, we focus on the first aspect: *preprocessing* of event logs. First, in Chapter 3, we describe a means of dealing with fine-granularity in event logs by defining abstractions of (low-level) events. Then, in Chapter 4, we advocate trace clustering as a means of dealing with heterogeneity in event logs. We propose context-aware approaches to trace clustering and show that considering contexts helps in improving process mining results. Finally, in Chapter 5, we propose techniques for dealing with process changes.

Chapter 3
Abstractions of Events

For process mining analysis, it would be ideal to have event logs emanating from well-structured processes at an appropriate level of abstraction. However, most real-life logs are far from ideal; the processes may not be well-structured (they can be flexible or less structured) and the events recorded can be too fine grained. This issue is more pronounced in high-tech system event logs as the logs are mostly generated by output statements that developers insert into the software supporting the system. Analysts and end users prefer a higher level of abstraction without being confronted with low level events stored in raw event logs. One of the major challenges in process mining is to bridge the gap between the higher level conceptual view of the process and the lower level implementation view.

In this chapter, we propose an event abstraction technique that exploits common execution patterns in event logs. This approach, inspired from techniques used in bioinformatics, comprises of

- *identifying common execution patterns* manifested in an event log
- *assessing the significance* of patterns and filtering insignificant ones
- *establishing relationships* between patterns and forming clusters of related patterns
- *defining abstractions* over the grouped patterns

In bioinformatics, a DNA sequence motif is defined as a nucleic acid sequence pattern that has some biological significance (both structural and functional) [48]. These motifs are usually found to recur in different genes or within a single gene. For example, tandem repeats (tandemly repeating DNA) are associated with various regulatory mechanisms such as protein binding [131]. More often than not, sequence motifs are also associated with structural motifs found in proteins thus establishing a strong correspondence between sequence and structure. Likewise, common subsequences of activities in an event log that are found to recur within a process instance or across process instances have some domain (functional) significance. We use some of the sequence patterns proposed in the string processing and bioinformatics literature [97, 98, 130, 131, 205] and adopt them to the process mining domain.

This chapter is organized as follows: Section 3.1 presents the workflow of a simple digital copier that we will use as a running example to explain the concepts in this thesis. The event logs simulated from this workflow will be used in our experiments throughout the thesis. Related work is presented in Section 3.2. In Section 3.3, we define a few interesting common execution patterns and correlate them to some of the process model constructs. Section 3.4 presents a few metrics that assist in assessing the significance of the patterns while Section 3.5 presents an approach to capture the relationships between patterns and create suitable abstractions. Section 3.6 presents and discusses some experimental results. We discuss the limitations of the proposed approach in Section 3.7. Finally, Section 3.8 concludes this chapter.

3.1 Running Example:- A Simple Digital Photo Copier

In order to illustrate the concepts in this thesis, we consider a simple digital photo copier as our running example. The copier supports photocopying, scanning, and printing of documents in both color and gray modes. The scanned documents can be sent to the user via email or FTP. Upon receipt of a job, the copier first generates an image of the document and subsequently processes the image to enhance its quality. Depending on whether the job request is for a copy/scan or print, dedicated procedures are used to generate an image. For print requests, the document is first interpreted and then a rasterization procedure is followed to form an image. The image is then written on the drum, developed, and fused on to the paper.

Figure 3.1 depicts the high-level workflow of the digital photo copier represented as a YAWL [224] model. This high-level workflow contains the composite tasks (subprocesses) Capture Image, Rasterize Image, Image Processing, and Print Image. Figure 3.2 depicts the Capture Image subprocess that is used to generate an image for copy or scan jobs while Figure 3.3 describes the Rasterize Image subprocess for generating an image of the documents submitted for printing. The Image Processing subprocess shown in Figure 3.4 performs some image quality enhancements and operations such as zooming and rotation on images. This subprocess contains a subprocess called Half Toning which is detailed in Figure 3.5. Figure 3.6 depicts the Print Image subprocess while the subprocesses Writing, Developing, and Fusing within Print Image are depicted in Figures 3.7, 3.8 and 3.9 respectively.

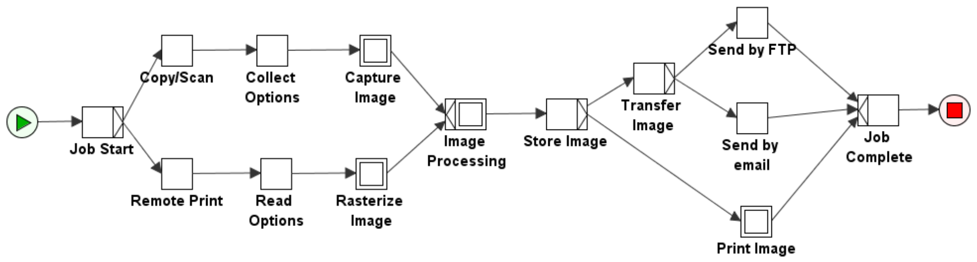


Figure 3.1: High-level model of the digital photo copier. The digital copier supports two functionalities, viz., copy/scan and print. Documents are interpreted and converted into an image before they are printed. Scanned images of the copy/scan jobs are sent to the user via email or ftp.

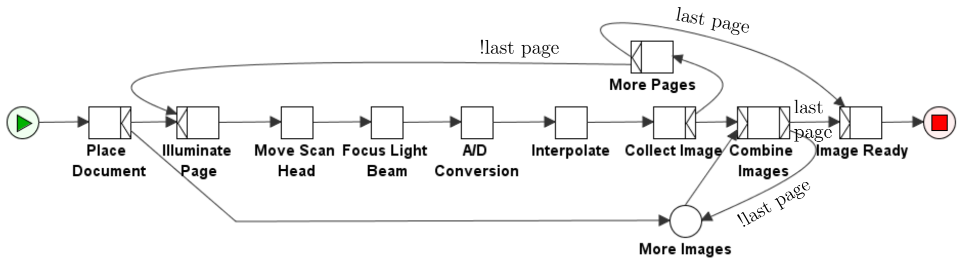


Figure 3.2: The capture image subprocess for copy or scan requests. Each page in the document is scanned separately and the scanned images are accumulated (combined) together. The scanning of each page is modeled using a loop construct. The combination of images can start as soon as the first page is scanned and can happen in parallel with the scanning of subsequent pages.

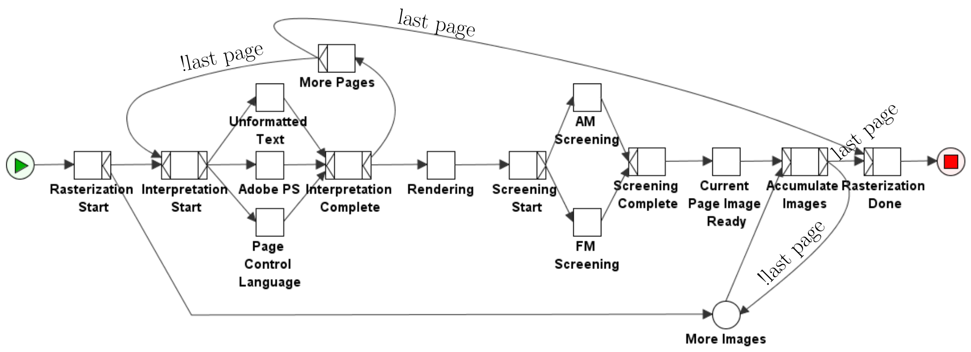


Figure 3.3: The rasterize image subprocess for print requests. Each page in the document is interpreted separately and rendered. The interpretation of each page is modeled using a loop construct. Three types of document interpretation are supported, viz., text, Adobe postscript and page control language (modeled using the XOR construct). The rendering can happen in parallel to the interpretation as soon as the first page is interpreted. The rendered images are accumulated (combined) together.

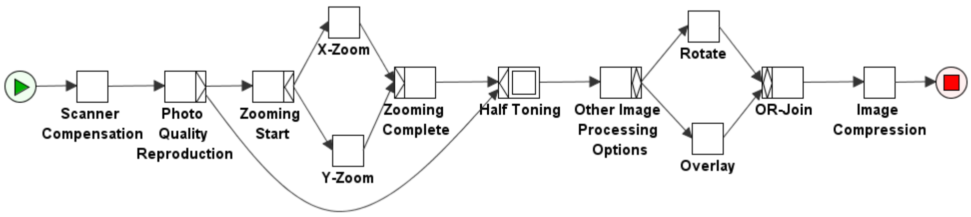


Figure 3.4: The image processing subprocess. This subprocess supports operations such as zooming, rotating, and overlay and attempts at improving the quality of images. The rotating and overlay functions are modeled using the OR (multi-choice) construct while the zooming functionality is modeled using the AND construct.

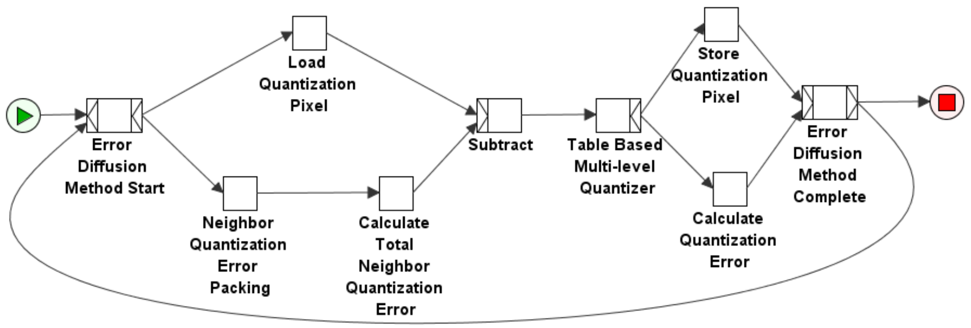

Load Quantization Pixel
Error Diffusion Method Start
Neighbor Quantization Error Packing
Calculate Total Neighbor Quantization Error
Subtract
Table Based Multi-level Quantizer
Store Quantization Pixel
Calculate Quantization Error
Error Diffusion Method Complete

Figure 3.5: The half-toning subprocess for image representation. Half toning is a technique that simulates continuous tone imagery through the use of equally spaced dots of varying size.

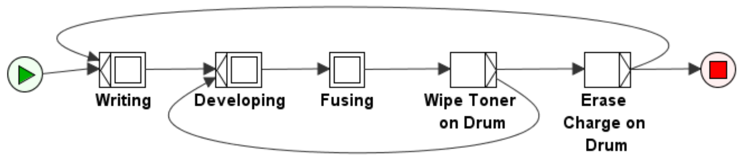


Figure 3.6: The print image subprocess for print job requests. The basic steps involved are the transfer of the image onto the drum (writing), application of toner (developing), pressing of the toner onto the paper (fusing), and cleaning the toner on the drum. The developing, fusing, and cleaning steps are repeated multiple times, once for each copy of the page. The while print image subprocess is executed as many times as the number of pages in the document.

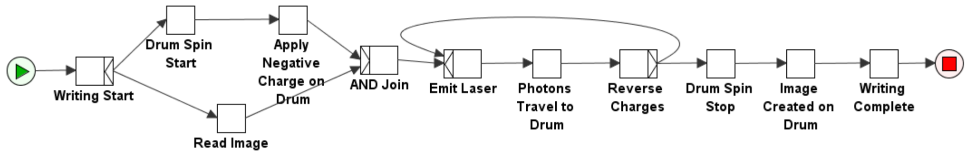


Figure 3.7: The writing subprocess within print image for writing the image on to the drum.

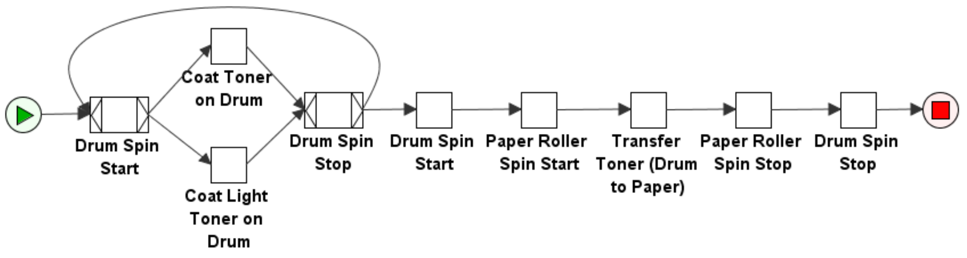


Figure 3.8: The developing subprocess within print image.

We have modeled this workflow of the copier in CPN tools [180] and generated event logs by simulation.

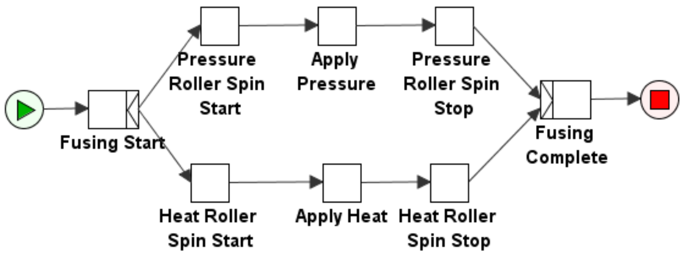


Figure 3.9: The fusing subprocess within print image.

3.2 Related Work

Several attempts have been reported in the literature on grouping events to create higher levels of abstraction. One such attempt is the use of semantic ontologies [29, 55]. The basic idea here is to define ontologies for the business and IT perspective and translate the tasks between the two models using reasoning techniques. Of specific interest is the EVents Ontology (EVO) [29] that captures the events taking place during the life-cycle of both business and IT processes at different levels of abstraction. The basis lies in the classification of events into management events, execution events, message events, process events, etc. Similarly, process models are linked to organizational information through organizational ontologies that capture concepts like organization, department, team or employee and the relationships between them. Although abstraction mechanisms based on semantics are interesting from a theoretical point of view, developing ontologies is a difficult task thereby making this approach less practical.

Another class of techniques pertains to the grouping of activities into clusters with each cluster defining an abstraction [96, 241, 254]. The activity-cluster association can be many-to-many in that an activity can belong to more than one cluster and a cluster can have multiple activities. In [254], this association is established by first creating a transition system based on the complete (or partial) prefix (or postfix) of the activities in a trace and then by synthesizing a process model (Petri net) from it using the well-known concept of the "theory of regions" [44, 58]. The theory of regions has a worst case complexity that is exponential in the size of the state space thereby making this approach less applicable in real-life logs. Van Dongen and Adriansyah [241] proposed an approach of mining a SPD model (Simple Precedence Diagram) wherein activities are grouped together based on their similarity using the Fuzzy k-Medoids algorithm [178]. The similarity between activities is defined based on the direct succession relationship between the events related by these activities in the event log. However, SPDs suffer from a few limitations, e.g., in the worst case, activities can be associated to all the clusters without any direct domain significance thereby making the interpretation of the clusters extremely hard. SPDs also lack guidelines to select a suitable number of clusters.

Günther et al. [96] proposed a means of forming hierarchical abstractions based on event correlations. The approach is based on the agglomerative hierarchical clustering technique [113] using the correlation of activities as a similarity metric. This results in a hierarchical model. Although the authors do not explicitly state it, they adopt a less restricted form of frequent episodes. Frequent episodes proposed by Manilla et al. [145] is a generic method to discover temporal patterns in time series data. An *episode* is a partially ordered set of events that occur relatively close to each other. The degree of closeness is guided by a *window* of fixed size provided by the user. Furthermore, the user needs to specify the number of windows in which an episode has to occur for it to be considered as frequent.

Window based techniques are most often used in episode mining and sequence mining. Even in process mining, classic algorithms such as the α-algorithm [229] uses the immediately follows relationship (window of length 2) to infer the control-flow between a pair of activities. Other techniques such as the Heuristics miner [264] and Fuzzy miner [94] use a larger window size as *look-ahead* to infer the control-flow between a pair of activities. These methods more often than not pick a *fixed length* window. The results are strongly influenced by the choice of the window length. The patterns considered in this chapter alleviate some of these problems. The length of the patterns is automatically determined by their contextual manifestation in the event log. Furthermore, the patterns capture the manifestation of commonly used control-flow constructs in process modeling and can be uncovered in *linear time* with respect to the size (total number of events) of the log. In contrast, the algorithms to discover frequent episodes have a *polynomial time* complexity with respect to the number of frequent episodes [145].

3.3 Common Execution Patterns

In this section, we define some of the sequence patterns and correlate them to the manifestation of process model constructs. We use these patterns as the basis for creating abstractions.

A *pattern* is a regularity in an event log. Such a regularity implies a sense of strong correlation between the elements (e.g., activities, resources, etc.) involved in the pattern and signify some interesting structures regarding a few aspects of the process rather than a summary about the whole process. In this thesis, we confine such regularities to sequence patterns (and not to partial orders like in frequent episodes [145]). Figure 3.10 depicts a few sequence patterns. Simple loops (loops over an individual activity or a sequence of activities) manifest as a repeated occurrence of an activity or subsequence of activities in *tandem* in the traces. The discovery of such manifestations in an event log would assist in identifying loop constructs in the process. Tandem repeats and tandem arrays [98] nicely capture such occurrences.

Definition 3.1 (Tandem Repeat). A *tandem repeat* (or square) in a sequence $\mathbf{s}$ is a subsequence $\mathbf{s}(i,j) = \alpha \diamond \alpha$ where α, called the *tandem repeat type*, is a non-empty sequence. Two occurrences of tandem repeats $\mathbf{s}(i,j) = \alpha \diamond \alpha$ and $\mathbf{s}(i',j') = \beta \diamond \beta$ are of the same type if and only if $\alpha = \beta$. ⌟

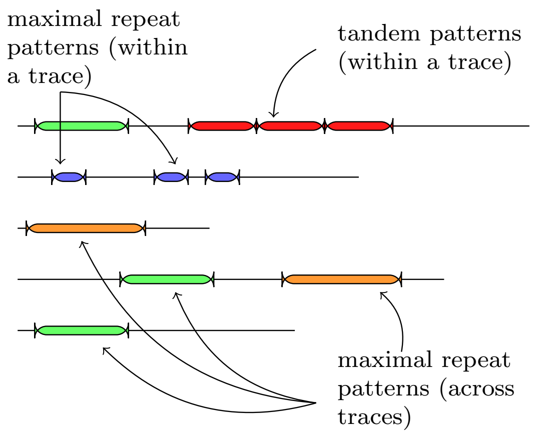

maximal repeat patterns (within a trace)
tandem patterns (within a trace)
maximal repeat patterns (across traces)

Figure 3.10: Common execution patterns.

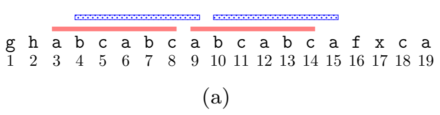
Definition 3.2 (Tandem Array). A *tandem array* in a sequence $\mathbf{s}$ is a subsequence $\mathbf{s}(i,j)$ of the form $\alpha \diamond \alpha \diamond \overset{k\ \text{times}}{\cdots} \diamond \alpha$, where α, called the tandem repeat type, is a non-empty sequence, and $k \geq 2$. We denote a tandem array by the triple (i, α, k) where the first element of the triple corresponds to the *starting position* of the tandem array, the second element corresponds to the *tandem repeat type*, and the third element corresponds to the *number of repetitions*. A tandem repeat is a special case of tandem array where the number of repetitions, k, is two. ⌟

Definition 3.3 (Maximal Tandem Array). A tandem array in a sequence $\mathbf{s}$, $\mathbf{s}(i,j) = \alpha \diamond \alpha \diamond \overset{k\ \text{times}}{\cdots} \diamond \alpha$ $(k \geq 2)$, is a *maximal tandem array* if and only if there are no additional copies of α immediately before or after $\mathbf{s}(i,j)$. ⌟

Definition 3.4 (Primitive Tandem Repeat Type). A tandem repeat type α is called a *primitive tandem repeat type* if and only if α is not a tandem array, i.e., there are no β and $k \geq 2$ such that $\alpha = \beta \diamond \beta \diamond \overset{k\ \text{times}}{\cdots} \diamond \beta$. ⌟

Definition 3.5 (Primitive Tandem Array). A tandem array $\mathbf{s}(i,j) = \alpha \diamond \alpha \diamond \overset{k\ \text{times}}{\cdots} \diamond \alpha$ $(k \geq 2)$, is a *primitive tandem array* if and only if α is a primitive tandem repeat type. ⌟

Consider a simple event log $\mathcal{L} = [$`ghabcabcabcabcafxca`, `abxcdxedfxgdxeh`, `bbbcdbbbccaa`, `abxcdxefygh`, `abxcfxgdxefdxeh`, `abxcdxefxgfygh`$]$. The trace $\mathbf{t}_1 =$ `ghabcabcabcabcafxca` contains a tandem repeat $\overline{\texttt{abcabc}}\ \overline{\texttt{abcabc}}$ with the tandem repeat type `abcabc` (see Figure 3.11(a)). Some of the tandem arrays in $\mathbf{t}_1$ are $(3, \texttt{abc}, 2)$, $(6, \texttt{abc}, 2)$, $(3, \texttt{abc}, 4)$, $(3, \texttt{abcabc}, 2)$, $(4, \texttt{bca}, 4)$, $(4, \texttt{bcabca}, 2)$, and $(5, \texttt{cab}, 3)$ (see Figure 3.11). The tandem repeat type `abcabc` is not a primitive tandem repeat type since it is a tandem array of `abc` with two repetitions. The tandem array $(3, \texttt{abc}, 2)$ is not a maximal tandem array because there exist four copies of `abc` starting at position 3 (see Figure 3.11(b)). Similarly, the tandem array $(6, \texttt{abc}, 2)$ is not a maximal tandem array because one copy of `abc` exists before and after the tandem array at position 6 (see Figure 3.11(b)). The tandem array $(3, \texttt{abcabc}, 2)$ is not a primitive tandem array because `abcabc` is not a primitive tandem repeat type: `abcabc` can be split into $\overline{\texttt{abc}}$ $\overline{\texttt{abc}}$.



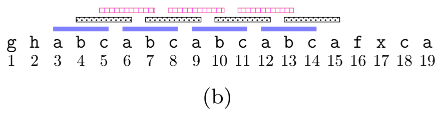


Figure 3.11: Tandem arrays in the trace $\mathbf{t}_1$ = ghabcabcabcabcafxca.

The maximal primitive tandem arrays in $\mathbf{t}_1$ are $(3, \texttt{abc}, 4)$, $(4, \texttt{bca}, 4)$, and $(5, \texttt{cab}, 3)$ while the ones in $\mathbf{t}_3$ are $(1, \texttt{b}, 3)$, $(6, \texttt{b}, 3)$, $(9, \texttt{c}, 2)$, and $(11, \texttt{a}, 2)$ (see Figure 3.12). The traces $\mathbf{t}_2$, $\mathbf{t}_4$, $\mathbf{t}_5$, and $\mathbf{t}_6$ do not contain any tandem arrays.

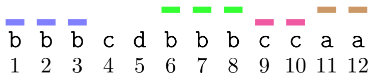


Figure 3.12: Tandem arrays in the trace $\mathbf{t}_3$ = bbbcdbbbccaa.

For any primitive tandem array (i, α, k), if $k \geq 3$, then there would be $|\alpha|$ primitive tandem arrays where the tandem repeat types are the cyclic permutations of α as illustrated in Figure 3.13. In the above example, the primitive tandem array $(3, \texttt{abc}, 4)$ in $\mathbf{t}_1$ led to two other tandem arrays $(4, \texttt{bca}, 4)$ and $(5, \texttt{cab}, 3)$ (see Figure 3.11(b)).

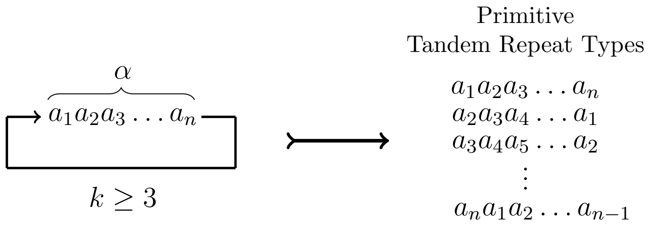


Figure 3.13: Redundant tandem repeat types owing to cyclic permutations.

Another class of patterns is formed by subsequences that do not manifest in tandem but nonetheless repeat within a single trace or across multiple traces. Such regularities signify some set of common functionality (often abstracted as a subprocess) accessed by the process. Regularities manifested within a single trace might indicate multiple invocations of a subprocess within the corresponding case. We consider maximal repeating subsequences as defined below.

Definition 3.6 (Maximal Pair). A *maximal pair* in a sequence $\mathbf{s}$ is a subsequence α that manifests in $\mathbf{s}$ at two distinct positions i and j such that the element to the immediate left (right) of the manifestation of α at position i is different from the element to the left (right) of the manifestation of α at position j, i.e., $\mathbf{s}(i, i+|\alpha|-1) = \mathbf{s}(j, j+|\alpha|-1) = \alpha$, and $\mathbf{s}(i-1) \neq \mathbf{s}(j-1)$ and $\mathbf{s}(i+|\alpha|) \neq \mathbf{s}(j+|\alpha|)$, for $1 \leq i < j \leq |\mathbf{s}|$

($\mathbf{s}(0)$ and $\mathbf{s}(|\mathbf{s}|+1)$ are considered to be null, i.e., $\langle\rangle$). A maximal pair is denoted by the tuple $((i,j),\alpha)$ where i and j correspond to the starting positions of the two manifestations of α in $\mathbf{s}$ with $i \neq j$. ⌟

Definition 3.7 (Maximal Repeat). A *maximal repeat* in a sequence $\mathbf{s}$ is defined as a subsequence α that occurs in a maximal pair in $\mathbf{s}$. ⌟

Figures 3.14(a)-(e) depict the maximal pairs in the trace $\mathbf{t}_1$ = `ghabcabcabcabcafxca`. The set of maximal repeats in this trace is {`a`, `ca`, `abca`, `abcabca`, `abcabcabca`}.

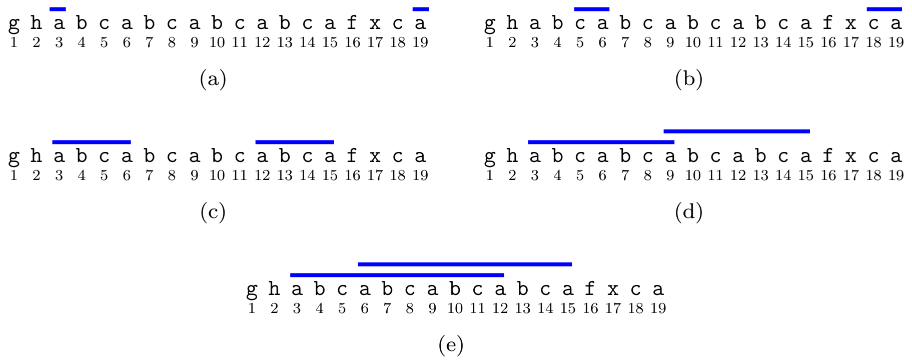


Figure 3.14: Maximal pairs in the trace $\mathbf{t}_1$ = `ghabcabcabcabcafxca`.

Definition 3.8 (Super Maximal Repeat). A *super maximal repeat* in a sequence is defined as a maximal repeat that never occurs as a proper subsequence of any other maximal repeat. ⌟

For the maximal repeats depicted in Figure 3.14, only the maximal repeat `abcabcabca` qualifies to be a super maximal repeat because all other maximal repeats happen to be a subsequence of `abcabcabca`.

Definition 3.9 (Near Super Maximal Repeat). A maximal repeat α in a sequence $\mathbf{s}$ is said to be a *near super maximal repeat* if and only if there exist at least one instance of α at some location in the sequence $\mathbf{s}$ where it is not contained in another maximal repeat. ⌟

The manifestation of the maximal repeat `ca` at position 18 in Figure 3.14(b) is not contained in any other maximal repeat manifestation. Hence the maximal repeat `ca` qualifies to be a near super maximal repeat. All instances of manifestation of the maximal repeats `a`, `abca`, and `abcabca` in Figures 3.14(a), (c), and (d) respectively are contained in the manifestations of the maximal repeats `ca` (for `a`) and `abcabcabca` (for `abca` and `abcabca`) in Figures 3.14(b) and (e). Hence, they do not qualify to be near super maximal repeats.

Near super maximal repeats are a hybrid between maximal repeats and super maximal repeats in that it contains all super maximal repeats and those maximal repeats that can occur in isolation in the sequence without being part of any other maximal repeat. Near super maximal repeats can assist in identifying potential

Table 3.1: Maximal, super maximal, and near super maximal repeats in each trace of the event log $\mathcal{L}$.

Id	Trace	Maximal Repeats	Super Maximal Repeats	Near Super Maximal Repeats
$\mathbf{t}_1$	ghabcabcabcabc-afxca	{a, ca, abca, abcabca, abcabcabca}	{abcabcabca}	{ca, abcabcabca}
$\mathbf{t}_2$	abxcdxedfxgdxeh	{d, x, dxe}	{dxe}	{d, x, dxe}
$\mathbf{t}_3$	bbbcdbbbccaa	{a, b, c, bb, bbbc}	{a, bbbc}	{a, c, bbbc}
$\mathbf{t}_4$	abxcdxefygh	{x}	{x}	{x}
$\mathbf{t}_5$	abxcfxgdxefdxeh	{f, x, dxe}	{f, dxe}	{f, x, dxe}
$\mathbf{t}_6$	abxcdxefxgfygh	{f, g, x}	{f, g, x}	{f, g, x}

choice constructs in the process model. Let us denote the set of maximal repeats, super maximal repeats, and near super maximal repeats by M, SM, and NSM respectively. The repeat sets satisfy the relation $SM \subseteq NSM \subseteq M$. The set $NSM \setminus SM$ (the set difference) depicts all maximal repeats that occur both in isolation and are also subsumed in some other maximal repeat. For any repeat $r \in NSM \setminus SM$, a super maximal repeat r_s which contains (subsumes) r can be either of the form $\alpha \diamond r$ or $r \diamond \beta$ or $\alpha \diamond r \diamond \beta$ (where α and β are subsequences of activities). This indicates that r can be a common functionality which might occur in conjunction with α and/or β. In other words, it indicates that α and β can potentially be optional (sequence of) activities in the context of r.

Table 3.1 depicts the maximal, super maximal, and near super maximal repeats present in each trace of the event log. The maximal repeat `abcabcabca` in $\mathbf{t}_1$ is not subsumed in any other maximal repeat and hence it qualifies to be a super maximal repeat. All other maximal repeats are subsumed in this maximal repeat. The maximal repeats {`ca`, `abcabcabca`} are near super maximal repeats because the last instance of `ca` at position 18 in $\mathbf{t}_1$ (see Figure 3.14(b)) is not contained in any other maximal repeat while both the instances of `abcabcabca` are not contained in any other maximal repeat.

Table 3.2 depicts the maximal, super maximal, and near super maximal repeats present in the entire event log, $\mathcal{L}$. In order to find the repeats across all the traces in the event log (as in Figure 3.10), we first construct a single sequence by concatenating all the traces in the event log with a *distinct* delimiter between the traces[1]. Repeats are then discovered in this concatenated sequence.

The patterns discussed above can all be uncovered using suffix trees. Suffix trees can be constructed in linear time with respect to the length of the sequence [214]. We can adopt efficient suffix tree construction techniques such as [35] to handle very

[1]For the example event log, the concatenated sequence corresponds to ghabcabcabcabcafxca$\perp_1$abxcdxedfxgdxeh$\perp_2$bbbcdbbbccaa$\perp_3$abxcdxefygh$\perp_4$abxcfxgdxefdxeh$\perp_5$ abxcdxefxgfygh (here, $\perp_i$'s (for $1 \leq i \leq 5$) are used as distinct delimiters between the traces)

Table 3.2: Maximal, super maximal, and near super maximal repeats across all traces in the event log, $\mathcal{L}$.

Maximal Repeat	{`a`, `b`, `c`, `d`, `f`, `g`, `h`, `x`, `ab`, `bb`, `bc`, `ca`, `cd`, `fx`, `gh`, `xc`, `dxe`, `fxg`, `abca`, `abxc`, `bbbc`, `dxef`, `dxeh`, `fygh`, `fxgdxe`, `abcabca`, `abxcdxe`, `abxcdxef`, `abcabcabca`}
Super Maximal Repeat	{`bbbc`, `dxeh`, `fygh`, `fxgdxe`, `abxcdxef`, `abcabcabca`}
Near Super Maximal Repeat	{`a`, `d`, `ca`, `cd`, `fx`, `gh`, `xc`, `fxg`, `abxc`, `bbbc`, `dxef`, `dxeh`, `fygh`, `fxgdxe`, `abxcdxe`, `abxcdxef`, `abcabcabca`}

long sequences. Gusfield and Stoye [98] proposed a linear time algorithm based on suffix trees to detect tandem repeats. Discovering tandem arrays takes $\mathcal{O}(n+z)$ time, where n is the length of the trace and z is the number of primitive tandem repeat types in the trace. Maximal, super maximal, and near super maximal repeats can be efficiently discovered in linear time using suffix trees [97, 130].

3.4 Assessing the Significance of Patterns

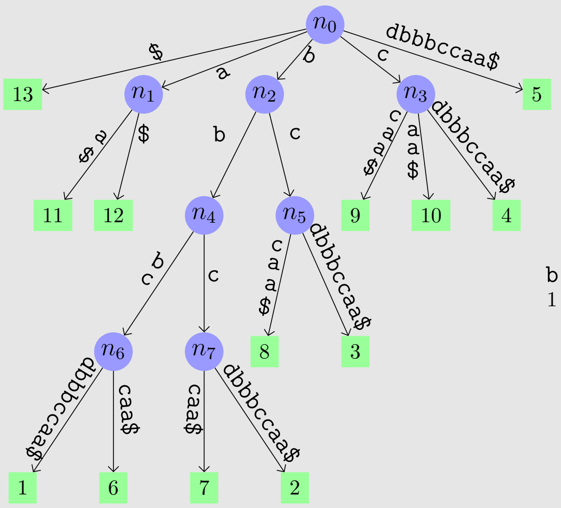
Large event logs and event logs defined over a large alphabet might contain abundant repeats. Not all of them might be characteristically significant. Henceforth, we refer to all recurring sequences (be it tandem repeat type or variants of maximal repeat) as a *pattern*. Metrics that assess the significance of patterns are required to filter out insignificant patterns. One of the fundamental measures is the *frequency of a pattern* (that estimates how often a pattern occurs in the event log). We need to consider only those patterns that appear much more often than they would be predicted to happen by chance alone. This measurement can be done in multiple ways. In this section, we define a few ways of estimating the frequency and thus the significance of a pattern.

We have seen earlier (see Figure 3.13) that primitive tandem arrays with at least three repetitions induce multiple tandem arrays that are cyclic permutations of the tandem repeat type sequence. Furthermore, a primitive tandem array (i, α, n) generates $n-1$ maximal repeats, $\alpha \diamond \alpha \diamond \overset{k \text{ times}}{\cdots} \diamond \alpha$, for $1 \leq k \leq n-1$, as illustrated in Figure 3.15. Similarly, manifestation of parallelism constructs can induce maximal repeats that are permutated sequences as illustrated in Figure 3.16. These can contribute significantly to the vast amount of uncovered patterns. We propose the use of pattern alphabets and the grouping together of patterns with the same pattern alphabet as a means of mitigating the abundance of patterns.

Definition 3.10 (Pattern Alphabet). Let $P_{\mathcal{L}}$ denote the set of all patterns uncovered in an event log $\mathcal{L}$. The pattern alphabet, $\Gamma(\mathbf{p})$, of a pattern, $\mathbf{p} \in P_{\mathcal{L}}$, is the set of activities that appear in $\mathbf{p}$. ⌟

Definition 3.11 (Equivalence Class of a Pattern Alphabet). The equivalence class of a pattern alphabet PA, is defined as $[\![PA]\!] = \{\mathbf{p} \mid \mathbf{p}$ is a pattern and $\Gamma(\mathbf{p}) =$

Maximal, super-maximal, and near-super-maximal repeats can be uncovered using suffix trees. The figure below depicts the suffix tree ST for the trace $\mathbf{t}$ = `bbbcdbbbccaa`. Recall from Section 2.4 that the leaves of a suffix tree represent the index position of suffixes in the trace. Let the *left symbol of a leaf* of ST denote the left symbol of the suffix position represented by that leaf, i.e., if a leaf contains a label i, then the left symbol of that leaf is $\mathbf{t}(i-1)$. A node v of ST is said to be *left diverse* if at least two leaves in v's subtree have different left symbols. The node n_1 is left diverse because the left symbols of its leaves 11 and 12 are `c` and `a` respectively. The node n_7 is not left diverse because the left symbol of both its leaves 7 and 2 is `b`. It is imperative that if a node v is left diverse then all its ancestors are left diverse as well. The left diverse nodes in the suffix tree are n_0, n_1, n_2, n_3, n_4, and n_6.



b	b	b	c	d	b	b	b	c	c	a	a	$
1	2	3	4	5	6	7	8	9	10	11	12	13

- *Maximal repeats* correspond to the path labels of all *left diverse* nodes in the suffix tree. Therefore, the maximal repeats are `a`, `b`, `c`, `bb`, and `bbbc`. There can be at most n maximal repeats in any sequence of length n.
- The path label corresponding to a left diverse internal node v in ST represents a *near-super-maximal repeat* if and only if one of v's children is a leaf and its left symbol is the left symbol of no other leaf of v. Only the left diverse nodes n_1, n_3, and n_6 satisfy this condition. Hence, the near super maximal repeats are `a`, `c`, and `bbbc`.
- The path label corresponding to a left diverse internal node v in ST represents a *super-maximal repeat* if and only if all of v's children are leaves and each has a distinct left symbol. Only the left diverse nodes n_1 and n_6 satisfy this condition. Therefore, the super maximal repeats are `a` and `bbbc`.

$PA\}$ ⌟

For example, for the patterns `abba`, `abdgh`, and `adgbh`, the pattern alphabets correspond to {`a`, `b`}, {`a`, `b`, `d`, `g`, `h`}, and {`a`, `b`, `d`, `g`, `h`} respectively. The equivalence class of the pattern alphabet {`a`, `b`, `d`, `g`, `h`} is $[\![\{\mathtt{a}, \mathtt{b}, \mathtt{d}, \mathtt{g}, \mathtt{h}\}]\!] = \{\mathtt{abdgh}, \mathtt{adgbh}\}$.

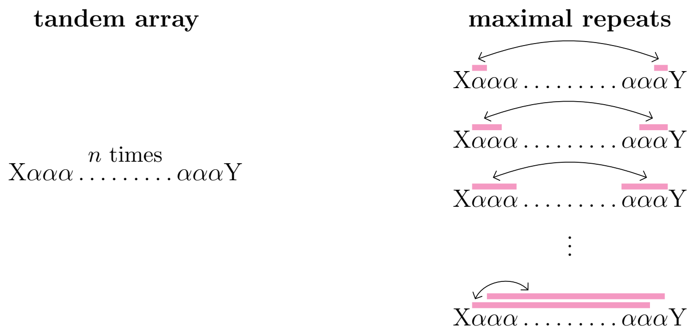


Figure 3.15: Tandem arrays lead to redundant maximal repeats. Every tandem repeat type α with n iterations that occur within the context of elements X and Y not in α leads to $n-1$ maximal repeats. The $n-1$ maximal repeats correspond to the k copies of α for $1 \le k \le n-1$. The k successive copies of α taken from either end form the maximal pairs because the left symbol of the sequence, $\alpha \diamond \alpha \diamond \cdots \diamond \alpha$ (k times), taken from the left end, X, differs from the left symbol, $\alpha(|\alpha|)$, of the sequence, $\alpha \diamond \alpha \diamond \cdots \diamond \alpha$ (k times), taken from the right end, (since X does not occur in α). Similarly, the right symbols also differ.

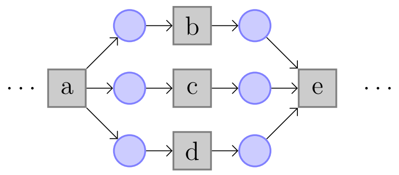


Figure 3.16: A process fragment with a parallelism construct. This parallelism construct can lead to manifestations such as `abcde`, `acdbe`, `abdce`, etc., in the event log, with a possibility for all or some of them to be uncovered as maximal repeats.

Definition 3.12 (Base Pattern). A *base pattern* is a pattern that does not contain any other pattern within it. In other words, a base pattern is one whose length is the same as that of the size of its alphabet. ⌟

The pattern `abba` is not a base pattern because it contains the patterns `a` (as a maximal repeat) and `b` (as a primitive tandem array as well as a maximal repeat) within it. In this case, the size of the pattern alphabet ($|\{\mathtt{a},\mathtt{b}\}| = 2$) is not equal to the length of the pattern ($|\mathtt{abba}| = 4$). The pattern `abdgh` is a base pattern.

The frequency or count of a pattern is a measure of how often it occurs in the event log. We define the pattern alphabet count to be the sum of counts of the patterns captured in its equivalence class. For example, consider the trace $\mathbf{t} = \mathtt{abxcdxedfxgdxeh}$ and the pattern alphabet equivalence classes $[\![\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{x}\}]\!] = \{\mathtt{abxc}\}$, $[\![\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{x}\}]\!] = \{\mathtt{abxcd}\}$, and $[\![\{\mathtt{d},\mathtt{e},\mathtt{x}\}]\!] = \{\mathtt{dxe},\mathtt{dxed}\}$. If we consider each of the patterns separately, the (pattern, frequency) pairs for the above trace correspond to $(\mathtt{abxc}, 1)$, $(\mathtt{abxcd}, 1)$, $(\mathtt{dxe}, 2)$, and $(\mathtt{dxed}, 1)$ (see Figure 3.17). The (pattern alphabet, frequency) pairs are $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{x}\}, 1)$, $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{x}\}, 1)$, and $(\{\mathtt{d},\mathtt{e},\mathtt{x}\}, 3)$. It is imperative to see that selected segments of the trace are contributing to more than one pattern (alphabet).

The activities in the subsequence $\mathtt{t}(5,7)$ contributed to two patterns, viz., `dxe` and `dxed` (see Figures 3.17(c) and (d)). Similarly, the activities in the subsequence $\mathtt{t}(1,4)$ contributed to two patterns, viz., `abxc` and `abxcd` (see Figures 3.17(a) and (b)). We call this method of computing pattern (alphabet) counts as *Overlapping Alphabet Count* (OAC). To define OAC we first need the definition of pattern indices, i.e., the positions where a pattern manifests in a trace.

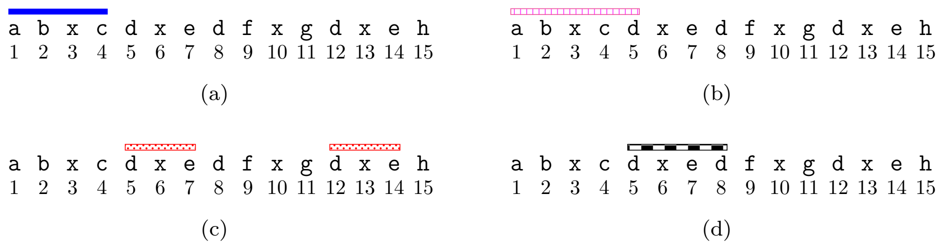


Figure 3.17: An example of overlapping alphabet counts.

Definition 3.13 (Pattern Indices). Given a pattern $\mathbf{p}$ and a trace $\mathbf{t}$ of length l (i.e., $|\mathbf{t}| = l$), the indices of the pattern $\mathbf{p}$ in trace $\mathbf{t}$ is defined as $PI(\mathbf{p}, \mathbf{t}) = \{i \mid 1 \leq i \leq l \ \wedge \ \mathbf{p}$ is a prefix of $\mathtt{t}(i,l)\}$. The indices of a set of patterns P in the trace $\mathbf{t}$ is defined as $PI(P, \mathbf{t}) = \bigcup_{\mathbf{p} \in P} PI(\mathbf{p}, \mathbf{t})$. ⌟

Definition 3.14 (Overlapping Alphabet Count (OAC)). Let PA be a pattern alphabet and $\mathcal{L}$ be an event log. The overlapping alphabet count of PA in $\mathcal{L}$ is defined as $OAC(PA, \mathcal{L}) = \sum_{\mathbf{t} \in \mathcal{L}} \mathcal{L}(\mathbf{t}) \sum_{\mathbf{p} \in [\![PA]\!]} |PI(\mathbf{p}, \mathbf{t})|$ ⌟

For example, the pattern indices sets of the patterns `dxe` and `dxed` in the trace $\mathbf{t} =$ `abxcdxedfxgdxeh` are $\{5, 12\}$ and $\{5\}$ respectively (see Figures 3.17(c) and (d)). The OAC of the pattern alphabet $\{\mathtt{d}, \mathtt{e}, \mathtt{x}\}$ in the trace is 3. Computing the pattern indices refers to the classic *exact string matching* problem in sequence mining [97]. Suffix trees can be used to efficiently compute the pattern indices and the overlapping count of a pattern. For a pattern $\mathbf{p}$ of length m and an event log or a trace of length n, this can be computed in $\mathcal{O}(n + m + k)$ where k denotes the number of occurrences of $\mathbf{p}$ in the event log or the trace ($\mathcal{O}(n)$ time to construct the suffix tree of the event log or the trace and $\mathcal{O}(m)$ time to preprocess the pattern). We can optimize this time for a set of patterns P. The trick lies in the fact that the event log or trace is fixed and we need to construct the suffix tree of the event log or trace only once and not for each pattern in the set. Thus all occurrences of each pattern $\mathbf{p} \in P$ can be found in $\mathcal{O}(m + k)$ time where m is the length of the pattern $\mathbf{p}$ and k is the number of occurrences of $\mathbf{p}$.

A pattern can be considered *significant* if it appears frequently often. The significance computed using overlapping alphabet counts may be misleading as regions in a trace can contribute to many patterns thus bloating their frequency. There is a need for restricting the kinds of occurrences to count when defining the frequency of a pattern alphabet. We define two such frequencies based on *non-overlapping* pattern counts, one considering non-overlap counts for each pattern alphabet separately (local) and the other considering non-overlap counts across all pattern alphabets (global) in the event log. Conflicts arise when more than one pattern can potentially contribute to the count at a region in a trace. One can assign preference to say, shorter (longer) patterns to resolve such conflicts. We define the shortest (longest) non-overlapping pattern at any index in a trace below.

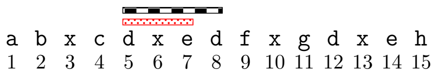
Definition 3.15 (Shortest and Longest Non-overlapping Pattern at Index). Let $\mathbf{t}$ be a trace and P be a set of patterns. The shortest non-overlapping pattern in P at index i in trace $\mathbf{t}$ is defined as $\overline{shortest}(P,\mathbf{t},i) = \{\mathbf{p} \in P \mid i \in PI(\mathbf{p},\mathbf{t}) \wedge \forall_{\substack{\mathbf{p}' \in P \\ \mathbf{p}' \neq \mathbf{p}}} (i \in PI(\mathbf{p}',\mathbf{t}) \Rightarrow |\mathbf{p}| < |\mathbf{p}'|)\}$.

Similarly, the longest non-overlapping pattern in P at index i in trace $\mathbf{t}$ is defined as $\overline{longest}(P,\mathbf{t},i) = \{\mathbf{p} \in P \mid i \in PI(\mathbf{p},\mathbf{t}) \wedge \forall_{\substack{\mathbf{p}' \in P \\ \mathbf{p}' \neq \mathbf{p}}} (i \in PI(\mathbf{p}',\mathbf{t}) \Rightarrow |\mathbf{p}| > |\mathbf{p}'|)\}$.

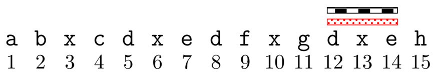
⌟

$\overline{shortest}(P,\mathbf{t},i)$ and $\overline{longest}(P,\mathbf{t},i)$ for $1 \le i \le |\mathbf{t}|$ is either empty or a singleton. Let $|\overline{shortest}(P,\mathbf{t},i)|$ and $|\overline{longest}(P,\mathbf{t},i)|$ denote the length of the shortest and longest patterns respectively. If no pattern exists at position i, then the length of the shortest (longest) pattern at index i is 0.

For example, consider the trace $\mathbf{t}$ = `abxcdxedfxgdxeh` and the set of patterns P = {`dxe`, `dxed`}. The shortest non-overlapping pattern at index 5 in trace $\mathbf{t}$ is `dxe` while the longest non-overlapping pattern at index 5 is `dxed` (see Figure 3.18(a)). The instance of `dxe` at index 12 corresponds to both the shortest and the longest non-overlapping pattern (see Figure 3.18(b)). The shortest and longest non-overlapping pattern sets correspond to the empty set for all indices other than 5 and 12 for this trace and pattern set.



(a) at index 5



(b) at index 12

Figure 3.18: An example of the shortest and longest patterns at an index.

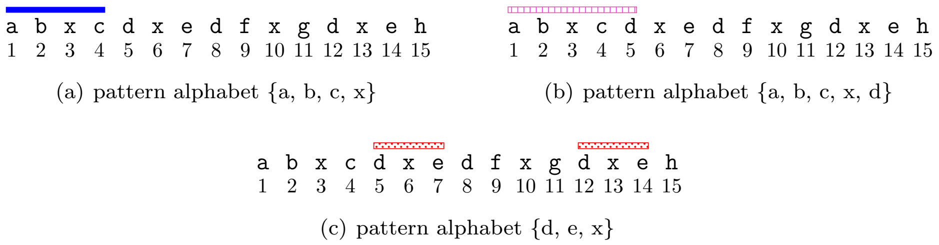
Definition 3.16 (Non-overlapping Pattern Indices). Given a set of patterns P and a trace $\mathbf{t}$, the non-overlapping pattern indices of the set of patterns in the trace, $NPI(P,\mathbf{t})$, is defined as

$$NPI(P,\mathbf{t}) = \{X \subseteq PI(P,\mathbf{t}) \mid \forall_{\substack{i,j \in X \\ j<i}} i \ge j + |\overline{shortest}(P,\mathbf{t},j)|\}.$$

Here, it is assumed that shorter patterns are preferred over longer patterns at any index j in the trace. If longer patterns are preferred, $\overline{shortest}(P,\mathbf{t},j)$ needs to be replaced with $\overline{longest(P, \mathbf{t}, j)}$ ⌟

Definition 3.17 (Non-overlapping Alphabet Count (NOAC)). Given a pattern alphabet PA and a trace $\mathbf{t}$, let $\widehat{NPI}([\![PA]\!], \mathbf{t})$ denote the maximal set of all non-overlapping pattern indices of the patterns under the equivalence class of the pattern alphabet, $[\![PA]\!]$, in the trace $\mathbf{t}$, i.e., $\widehat{NPI}([\![PA]\!], \mathbf{t})$ is an $X \in NPI([\![PA]\!], \mathbf{t})$ such that $X = \arg\max_{X' \in NPI([\![PA]\!], \mathbf{t})} |X'|$. The non-overlapping alphabet count of a pattern alphabet PA in the event log $\mathcal{L}$ is defined as $NOAC(PA, \mathcal{L}) = \sum_{\mathbf{t} \in \mathcal{L}} \mathcal{L}(\mathbf{t}) |\widehat{NPI}([\![PA]\!], \mathbf{t})|$ ⌟

Consider again the trace $\mathbf{t} = \texttt{abxcdxedfxgdxeh}$ and the pattern alphabets with equivalence classes $[\![\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{x}\}]\!] = \{\texttt{abxc}\}$, $[\![\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{x}\}]\!] = \{\texttt{abxcd}\}$, and $[\![\{\texttt{d}, \texttt{e}, \texttt{x}\}]\!] = \{\texttt{dxe}, \texttt{dxed}\}$. The NOAC (with preference to shorter patterns) for given example is $(\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{x}\}, 1), (\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{x}\}, 1)$, and $(\{\texttt{d}, \texttt{e}, \texttt{x}\}, 2)$ (see Figures 3.19(a)–(c)). Note that the conflict at position 5 in $\mathbf{t}$ for pattern alphabet $\{\texttt{d}, \texttt{e}, \texttt{x}\}$ is resolved in favor of the pattern dxe thereby making $\mathbf{t}(5, 7)$ contribute to only one pattern (see Figure 3.19(c)).



(a) pattern alphabet {a, b, c, x}

(b) pattern alphabet {a, b, c, x, d}

(c) pattern alphabet {d, e, x}

Figure 3.19: An example of non-overlapping pattern alphabet counts.

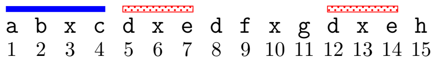
Though NOAC addresses the contribution of a region in a trace to only one of the patterns in the equivalence class of a given pattern alphabet, it still might lead to situations where a region contributes to more than one pattern when a set of pattern alphabets is considered. For example, the activities in the subsequence at $\mathbf{t}(1, 5)$ contributed to all the three pattern alphabets in the above example: $\mathbf{t}(1, 4)$ to $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{x}\}$, $\mathbf{t}(1, 5)$ to $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{x}\}$ and $\mathbf{t}(5)$ to $\{\texttt{d}, \texttt{e}, \texttt{x}\}$. In order to alleviate this problem, we consider the non-overlapping global alphabet counts (global over a set of pattern alphabets). Let $\mathbb{PA}_G$ be a set of pattern alphabets (e.g., set of all pattern alphabets in an event log) and let P_G denote the set of all patterns defined by $\mathbb{PA}_G$, i.e., $P_G = \bigcup_{PA \in \mathbb{PA}_G} [\![PA]\!]$.

Definition 3.18 (Non-overlapping Global Alphabet Count (NOGAC)). Given a global set of pattern alphabets $\mathbb{PA}_G$, a global set of patterns P_G defined by it, and a pattern alphabet $PA \in \mathbb{PA}_G$, let $\widehat{NPI_G}([\![PA]\!], \mathbf{t}, P_G)$ denote the maximal set of all non-overlapping pattern indices of the patterns under the equivalence class of the pattern alphabet PA in the trace $\mathbf{t}$ with respect to the global pattern set P_G and is defined as

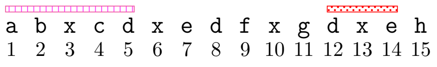
$$\widehat{NPI_G}([\![PA]\!], \mathbf{t}, P_G) = \{i \in \widehat{NPI}(P_G, \mathbf{t}) \mid \overline{shortest(P_G, \mathbf{t}, i)} \in [\![PA]\!]\}.$$

The non-overlapping global alphabet count of the pattern alphabet PA in an event log $\mathcal{L} : \mathbb{T} \rightarrow \mathbb{N}_0$ is defined as $NOGAC(PA, \mathcal{L}, P_G) = \sum_{\mathtt{t}\in\mathcal{L}} \mathcal{L}(\mathtt{t})|\widehat{NPI}_G([\![PA]\!], \mathtt{t}, P_G)|$ ⌟

For the above example, the NOGAC across all alphabets with preference to shorter patterns is $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{x}\},1), (\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{x}\},0)$, and $(\{\mathtt{d},\mathtt{e},\mathtt{x}\},2)$ (see Figure 3.20(a)) while the NOGAC with preference to longer patterns is $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{x}\},0), (\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{x}\},1)$, and $(\{\mathtt{d},\mathtt{e},\mathtt{x}\},1)$ (see Figure 3.20(b)).



(a) preference to shorter patterns



(b) preference to longer patterns

Figure 3.20: An example of non-overlapping global alphabet count.

Definition 3.19 (Conservedness (CON)). The *conservedness* of a pattern alphabet PA, measures the degree to which the individual activities involved in the pattern alphabet manifest as the patterns defined by the alphabet and is defined as $CON(PA, \mathcal{L}) = \frac{NOAC(PA,\mathcal{L})}{\mu}\left(1 - \frac{\sigma}{\mu}\right) * 100\%$ where μ and σ denote the mean and standard deviation of the frequency of activities defined by PA in the event log $\mathcal{L}$[2]. ⌟

The factor $\frac{\sigma}{\mu}$ is the *coefficient of variation* and measures the dispersion of the distribution of activities in the log. For example, consider three pattern alphabets {a,b,c}, {a,b,c,d}, and {a,b,c,e}. Let the equivalence classes of these pattern alphabets be {abc}, {abcd}, and {abce} respectively. Let the non-overlap alphabet count of these pattern alphabets be $(\{\mathtt{a},\mathtt{b},\mathtt{c}\},100)$, $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\},60)$, and $(\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{e}\},40)$ and the frequency of activities be $(\mathtt{a},100)$, $(\mathtt{b},100)$, $(\mathtt{c},100)$, $(\mathtt{d},60)$, and $(\mathtt{e},40)$. The mean frequencies of the activities involved in the pattern alphabets {a, b, c}, {a, b, c, d}, and {a, b, c, e} are 100, 90, and 85 respectively while the standard deviations correspond to 0, 20, and 30 respectively. It is to be noted that not all occurrences of a, b, c manifest as abcd (or abce); however, all occurrences of a, b, c manifest as abc. Hence, the conservedness values of the pattern alphabets {a, b, c}, {a, b, c, d}, and {a, b, c, e} are $\frac{100}{100}(1 - \frac{0}{100}) * 100 = 100\%$, $\frac{60}{90} * (1 - \frac{20}{90}) * 100 = 51.8\%$, and $\frac{40}{90} * (1 - \frac{30}{90}) * 100 = 29.6\%$ respectively.

Concurrency in process models affects the manifestation of patterns. For example, let us assume that a sequence involving two activities a and b occuring in parallel with another branch. One can notice the pattern ab manifested in some of the traces in the event log. In other traces, the activities involved in the parallel subprocess might manifest in between a and b thus making the $NOAC$ of the pattern alphabet {a,b} (with ab as the pattern under its equivalence class) considerably lower when compared to their mean count. Accordingly the conservedness value would be low for this pattern alphabet. However, it is to be noted that a and b are highly correlated

[2]Although we have defined conservedness using only the $NOAC$, one can define variants by substituting $NOAC$ with other count metrics, viz., OAC and $NOGAC$.

with $\sigma = 0$. In order to compensate for pattern manifestations affected due to concurrency in process models, we introduce a weighted variant of conservedness, $WCON$, defined as $WCON(PA, \mathcal{L}) = \left(1 - \frac{\sigma}{\mu}\right) * \left(\eta + (1-\eta)\frac{NOAC(PA,\mathcal{L})}{\mu}\right) * 100\%$ where $0 \leq \eta \leq 1$.

We recommend that pattern alphabets with a high *(weighted)conservedness* value be considered as candidates for abstraction.

One can also consider the percentage of instances in which a pattern is manifested as a measure of assessing the significance of a pattern. The instance percentage of a pattern alphabet is defined as in Definition 3.20.

Definition 3.20 (Instance Percentage). The instance percentage of a pattern alphabet PA in an event log $\mathcal{L}$, $IP(PA, \mathcal{L})$, is the percentage of instances in which at least one pattern defined by the equivalence class of the pattern alphabet is manifested. In other words, let $\mathcal{L}_{PA} \subseteq \mathcal{L}$ be the set of all traces $\mathtt{t} \in \mathcal{L}$ such that $PI(PA, \mathtt{t}) \neq \emptyset$. $IP(PA, \mathcal{L}) = 1/|\mathcal{L}| \sum_{\mathtt{t} \in \mathcal{L}_{PA}} \mathcal{L}(\mathtt{t}) \times 100\%$ where $\mathcal{L}(\mathtt{t})$ is the multiplicity of the trace $\mathtt{t}$ in $\mathcal{L}$ and $|\mathcal{L}|$ is the total number of traces in $\mathcal{L}$. ⌟

3.5 Capturing Relationships Between Patterns

Relationships exist between patterns. For example, consider the patterns `dxefxg`, `dxe`, and `fxg`. It could be the case that `dxe` and `fxg` are *sub-functionalities* used also in a larger context `dxefxg`. As another example, consider the patterns `abcd` and `abd`. It is most likely for activity `c` to represent a functionality similar to that of `a`, `b`, and `d`, since `c` occurs within the context of `a`, `b`, and `d`. Subprocess abstractions can be defined by considering a *partial ordering* on the pattern alphabets. We use *subsumption* as the *cover* relation. By defining a partial order on the pattern alphabets, one can construct a *Hasse* diagram. The Hasse diagram, henceforth referred to as a *pattern graph*, over pattern alphabets can be used as a means for defining abstractions of events. Figure 3.21 depicts the pattern graph over the pattern alphabets defined by the *base* maximal repeats in Table 3.2. The pattern alphabets $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{x}\}, \{\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{x}\}, \{\mathtt{d},\mathtt{e},\mathtt{h},\mathtt{x}\}, \{\mathtt{f},\mathtt{g},\mathtt{h},\mathtt{y}\}$, and $\{\mathtt{f},\mathtt{g},\mathtt{x}\}$ are the *maximal elements* of the partial ordering.

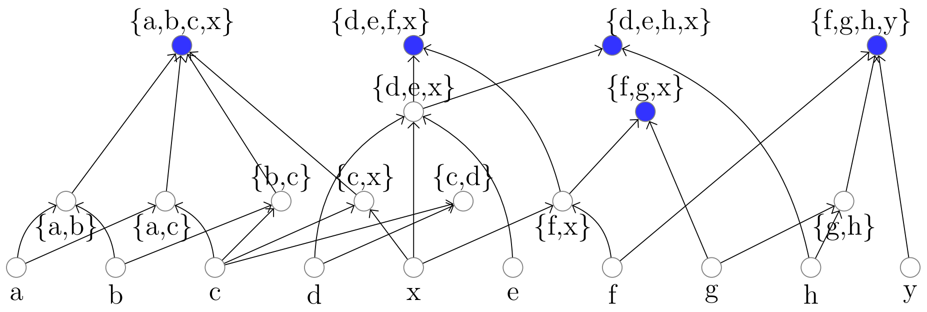


Figure 3.21: A pattern graph. Shaded nodes represent the maximal elements of the pattern graph.

Nodes (signifying pattern alphabets) in a pattern graph, which capture common execution patterns in an event log, form the basis for abstraction. One can use these abstractions to preprocess the event log and transform it into an event log with higher levels of abstraction. The basic idea is to replace all manifestations of patterns defined by the pattern alphabets with the abstract activity defined for the pattern alphabet. In this chapter, we look at only the criteria for selection of nodes for abstraction and the basic idea on how to utilize them for transforming logs. The transformation technique will be presented in detail in Chapter 6. Two types of node selection modes can be considered:

- **Single Node Mode:** All manifestations of patterns under the equivalence class of this node's pattern alphabet are represented by the same abstract activity in the transformed log.
- **Sub-graph Mode:** All manifestations of patterns under the equivalence class of pattern alphabets defined by the induced subgraph at any selected node are substituted by the abstract activity of the selected node during transformation. For example, one can consider the maximal elements and its induced sub-graph in the pattern graph as a candidate for abstraction.

For example, let us assume that the pattern alphabets $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$, $\{\mathtt{d}, \mathtt{e}, \mathtt{h}, \mathtt{x}\}$, and $\{\mathtt{f}, \mathtt{g}, \mathtt{x}\}$ in Figure 3.21 are chosen for abstractions. Let the pattern alphabet equivalent classes corresponding to these pattern alphabets be $\llbracket\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}\rrbracket = \{\mathtt{abxc}\}$, $\llbracket\{\mathtt{d}, \mathtt{e}, \mathtt{h}, \mathtt{x}\}\rrbracket = \{\mathtt{dxeh}\}$, and $\llbracket\{\mathtt{f}, \mathtt{g}, \mathtt{x}\}\rrbracket = \{\mathtt{fxg}\}$. Let W, V, and U respectively be the abstract activities chosen for these three pattern alphabets. If these pattern alphabets are considered under the single node mode, then only the patterns defined by these alphabets are considered. Consider the two traces $\mathbf{t}_1 = \mathtt{abxcdxedfxgdxeh}$ and $\mathbf{t}_2 = \mathtt{abxcdxefxg}$. Figure 3.22(a) highlights the traces with the pattern manifestations and Figure 3.22(b) depicts the transformed traces after replacing the pattern manifestations with their corresponding abstract activities, e.g., since the pattern alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$ is associated with the abstract activity W, the pattern manifestation abxc is replaced with W.

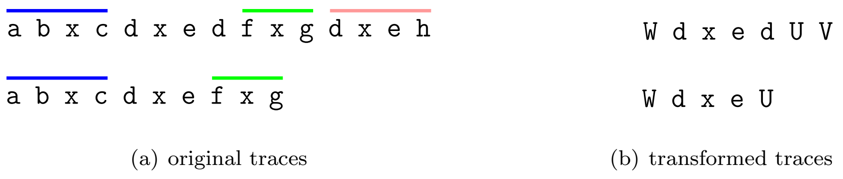
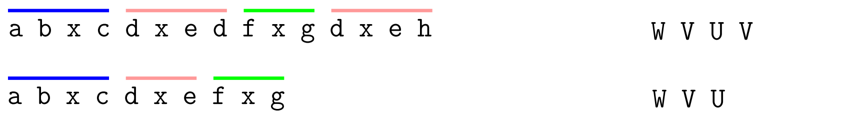


Figure 3.22: Transformation of traces with abstractions based on single node selection mode.

However, if the pattern alphabets are considered under the sub-graph mode, we also consider all the pattern alphabets subsumed under the three pattern alphabets, e.g., $\{\mathtt{d}, \mathtt{e}, \mathtt{x}\}$, $\{\mathtt{d}\}$, $\{\mathtt{e}\}$, and $\{\mathtt{x}\}$ subsumed under the pattern alphabet $\{\mathtt{d}, \mathtt{e}, \mathtt{h}, \mathtt{x}\}$. Let dxe and dxed be the patterns under the equivalent class of the pattern alphabet

$\{\mathtt{d}, \mathtt{e}, \mathtt{x}\}$, i.e., $\llbracket\{\mathtt{d}, \mathtt{e}, \mathtt{x}\}\rrbracket = \{\mathtt{dxe}, \mathtt{dxed}\}$. Now the pattern manifestations of all pattern alphabets in the sub-graph of the selected node are processed and replaced with its abstract activity. Figure 3.23(a) highlights the traces with the pattern manifestations and Figure 3.23(b) depicts the transformed traces after replacing the pattern manifestations with their corresponding abstract activities. Unlike the single node mode, the pattern manifestation `dxed` in $\mathtt{t}_1$ and the pattern manifestation `dxe` in $\mathtt{t}_2$ are also processed under the sub-graph mode because their pattern alphabet {`d`, `e`, `x`} is subsumed in {`d`, `e`, `h`, `x`}.



(a) original traces (b) transformed traces

Figure 3.23: Transformation of traces with abstractions based on sub-graph selection mode.

When considering abstractions using the sub-graph mode, there could be scenarios where certain pattern alphabets contribute to more than one abstraction. Such pattern alphabets can either be put in one of the abstractions (based on their contexts of manifestation in the event log) or can define an abstraction in itself. Such alphabets can be considered as a (sub-)functionality that is used in different contexts. For example, assuming that the maximal elements are chosen to define abstractions using the sub-graph mode, the pattern alphabet {`f`, `x`} is subsumed in two maximal elements {`f`, `g`, `x`} and {`d`, `e`, `f`, `x`}.

In order to reduce the total number of abstract activities introduced, one can define extended joins on the maximal elements. There can be instances where two maximals of the partial ordering on the pattern alphabets have a lot of activities in common, e.g., the maximal elements {`d`, `e`, `f`, `x`} and {`d`, `e`, `h`, `x`}. It could be the case that `f` and `h` are activities under a choice construct within the context of `d`, `e`, and `x`. The two maximal elements can be extended to join at {`d`, `e`, `f`, `h`, `x`} as depicted in Figure 3.24. Different criteria for extending the maximal elements can be defined. Let PA_1^m and PA_2^m be the pattern alphabets corresponding to two maximal elements in a pattern graph; one can choose to extend two maximal elements provided they share a set of common elements above a particular threshold and also when the differences between them is less. In other words, extend the maximal elements only if $|PA_1^m \cap PA_2^m| \geq \delta_c$ and $|(PA_1^m \setminus PA_2^m) \cup (PA_2^m \setminus PA_1^m)| \leq \delta_d$. δ_c corresponds to the threshold on the number of common elements which can either be a fixed constant or a fraction of the cardinality of the participating maximal elements such as $0.75 \times \min(|PA_1^m|, |PA_2^m|)$. δ_d corresponds to the threshold on the number of differences between the two maximal elements.

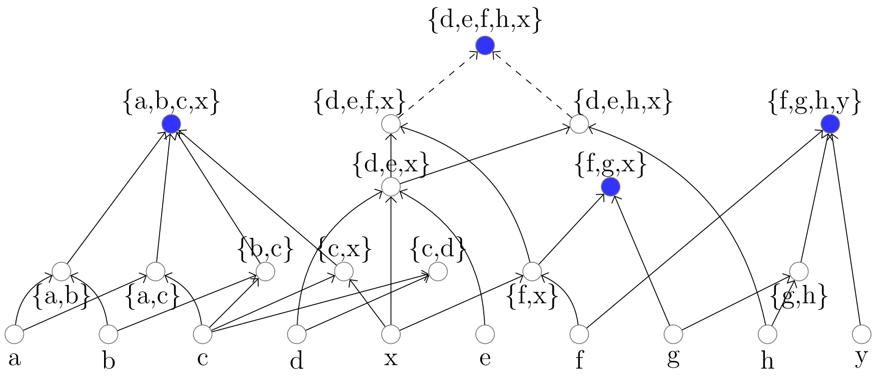


Figure 3.24: A pattern graph with extended joins (indicated by dashed lines). Shaded nodes represent the maximal elements of the pattern graph.

3.6 Experiments and Discussion

In this section, we present some of the patterns and abstractions uncovered using an event log of the simple digital copier example. As described in Section 3.1, we have modeled the workflow of the copier in CPN tools [180] and generated event logs using simulation. We consider one such event log for our analysis. The event log consists of 100 cases, 40, 995 events, and 76 event classes (distinct activities). As specified in the YAWL models the workflow of the copier contains abstract activities (composite tasks) in the form of subprocesses (see Figures 3.1, 3.4, and 3.6) whereas the simulated event log contains low level events. An ideal output would be to uncover the abstractions with domain significance as specified by the abstract activities in the YAWL models. We have implemented the discovery of patterns, computation of pattern metrics, and the determination of abstractions as the Pattern Abstractions plug-in in ProM (cf. Chapter 10).

We first identify the tandem arrays in the event log. Table 3.3 depicts the pattern alphabets corresponding to the maximal primitive tandem arrays in the event log. Pattern alphabets 1 and 2 correspond to the loop construct in the Developing subprocess (see Figure 3.8). Pattern alphabet 3 corresponds to the loop construct in the Writing subprocess (see Figure 3.7). Pattern alphabets 4 and 5 pertain to the Rasterize Image subprocess (see Figure 3.3). This loop construct is not explicit in this subprocess. The rasterize image subprocess is invoked for print job requests and each page in the document submitted for print is first interpreted, rendered, screened, and converted into an image. The rendering and screening can start as soon as an interpreted page is available and subsequently can be done in parallel to interpretation. In the copier, rendering and screening takes more time than the interpretation. Hence, there could be scenarios where all the pages have been interpreted but some of them yet to be rendered and screened. In such cases, the rendering and screening being the only tasks left manifest as tandem arrays in the event log. This is what is uncovered in this pattern alphabet. Pattern alphabets 6 and 7 correspond to the Capture image (see Figure 3.2) and Half Toning subprocesses (see Figure 3.5).

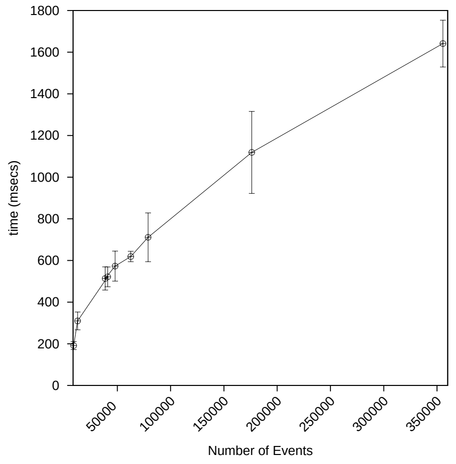
Table 3.3: Pattern alphabets corresponding to the maximal primitive tandem arrays in the simple digital copier event log.

S.No	Pattern Alphabet	Abstraction
1	{Drum Spin Start, Coat Toner on Drum, Drum Spin Stop}	Coat Toner
2	{Drum Spin Start, Coat Light Toner on Drum, Drum Spin Stop}	
3	{Emit Laser, Photons Travel to Drum, Reverse Charges}	Charge Drum
4	{Rendering, Screening Start, FM Screening, Screening Complete, Current Page Image, Accumulate Images}	Render and Screen
5	{Rendering, Screening Start, FM Screening, Screening Complete, Current Page Image, Accumulate Images}	
6	{Illuminate Page, Move Scan Head, Focus Light Beam, A/D Conversion, Interpolate, Filtered Image, Collect Image}	Capture Image
7	{Error Diffusion Method Start, Load Quantization Pixel, Neighbor Quantization Error Packing, Calculate Total Neighbor Quantization Error, Subtract, Table Based Multi-level Quantizer, Store Quantization Pixel, Calculate Quantization Error, Error Diffusion Method Complete }	Half Toning

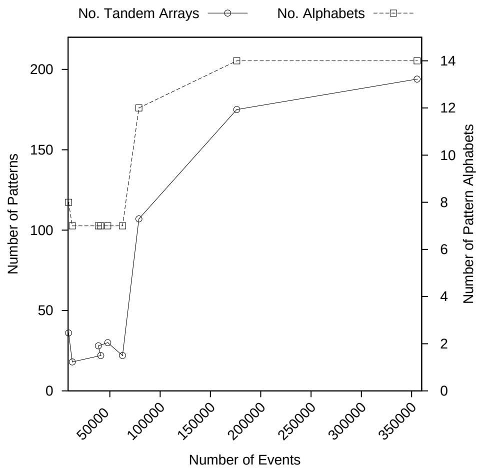
Let us now consider these pattern alphabets and form abstractions. Since none of the pattern alphabets in Table 3.3 subsumes another one, all seven pattern alphabets are maximal elements in the pattern graph. However, the pattern alphabets 1 and 2 and the pattern alphabets 4 and 5 share a majority of activities between them and can be merged. Thus we have 5 maximal elements as candidates for abstraction. One can give meaningful names to these abstractions as depicted in Table 3.3, e.g., ({1,2}, Coat Toner), ({3}, Charge Drum), ({4,5}, Render and Screen), ({6}, Capture Image), and ({7}, Half Toning). It is important to note that the abstractions being based on common execution patterns have a strong domain significance from a functionality point of view. Likewise, one can consider the maximal repeat patterns to define abstractions.

One can use the approach presented in this chapter in an iterative way to form multiple levels of abstraction. The basic idea is to uncover the patterns and determine abstractions from an event log and use these abstractions to transform the event log. In the transformation process (we will look at it in detail in Chapter 6), the manifestations of patterns defined by the pattern alphabets involved in the abstraction are replaced with the abstract activity. The transformed log can then be used as the input for the next iteration and this process can be repeated.

In the next two subsections, we discuss on the scalability of the approach and report the influence of the size of the log (total number of events) and the size of the alphabet on pattern discovery.



(a) average computation time to discover all tandem arrays



(b) the number of tandem arrays and pattern alphabets pertaining to the primitive tandem repeat types

Figure 3.25: The number of tandem arrays and the amount of time to uncover them for varying log sizes. The 95% confidence intervals are also indicated.

3.6.1 Influence of Log Size

We have simulated multiple event logs of the digital copier example with varying number of cases (and thereby the total number of events). Figure 3.25(a) depicts the average computational time along with the 95% confidence intervals (over five independent runs) taken to discover all tandem arrays for different event logs[3]. We can see that the time required to uncover all tandem arrays varies linearly with respect to the size of the event log as expected. Figure 3.25(b) depicts the number of distinct primitive tandem repeat types and pattern alphabets over the different logs. The notable difference between the number of patterns and the pattern alphabets can be attributed to the cyclic permutations of the tandem repeat types (cf. Section 3.3 and Figure 3.13). We can see that after a certain size of the log, the number of pattern alphabets stabilizes indicating a *sense of completeness* of the log with respect to the loop construct manifestations. Figure 3.26 depicts the average computation time along with the 95% confidence intervals (over five independent runs) required to compute the overlapping and non-overlapping counts of the tandem repeat patterns for varying sizes of the event log. Estimating the counts varies linearly with respect to the size of the event log and the number of distinct patterns.

Figure 3.27(a) depicts the average computation time along with the 95% confidence interval (over five independent runs) required to construct a suffix tree for varying sizes of the event log. The suffix tree is constructed on the sequence obtained by

[3]All the computation times reported in this chapter are measured on an i3 Core CPU M350 @ 2.27 GHz with 4GB RAM running a 64-bit Windows 7 OS.

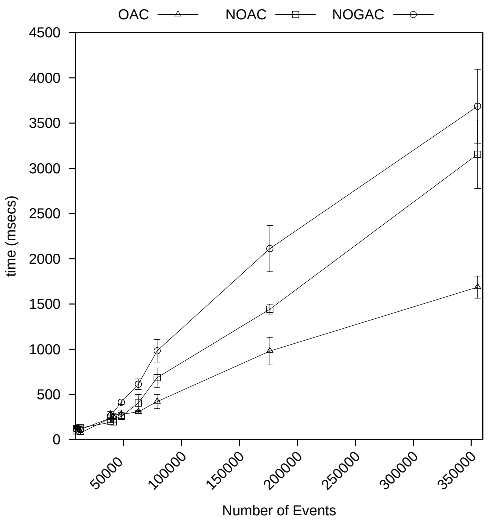
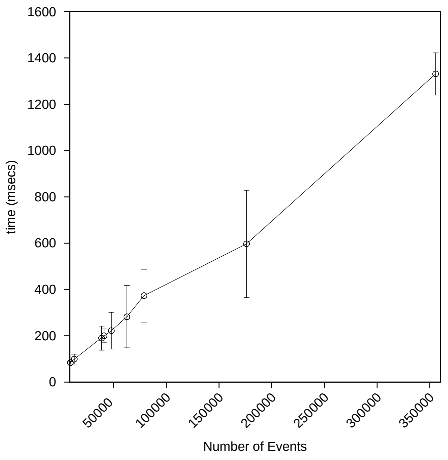


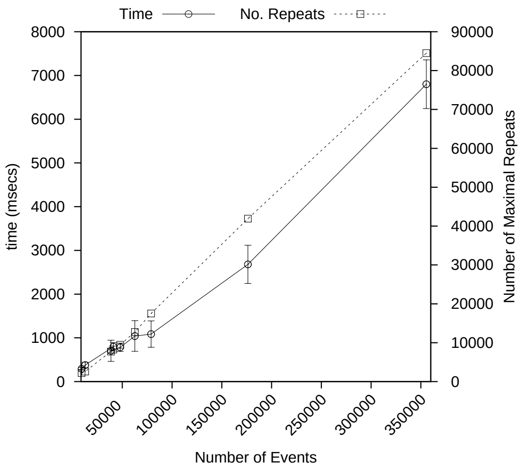
Figure 3.26: Computational time required to estimate the various count metrics for the tandem repeat patterns. The 95% confidence intervals are also indicated.

concatenating all the traces with a *distinct* delimiter between them. Recall that this suffix tree is necessary to discover the maximal repeats in an event log. Figure 3.27(b) depicts the average time and 95% confidence intervals (over five independent runs) required to compute all maximal repeats in these event logs. As expected, we can see that the construction time of suffix tree varies linearly with respect to the length of the sequence (size of the event log) while the uncovering of maximal repeats varies linearly with respect to the number of distinct maximal repeats, which in turn is linear to the size of the event log. At first inspection one might be alarmed by the number of maximal repeats uncovered. However, it is to be noted that a lot of these maximal repeats contain redundant information (recall from Figures 3.15 and 3.16).

Figure 3.28(a) depicts the average time along with the 95% confidence intervals (on five independent runs) required to compute the various count metrics over the maximal repeat patterns. The computation time is linear to the size of the event log and the number of patterns and the computation of different count metrics takes almost the same time. Also indicated in Figure 3.28(a) are the number of pattern alphabets and the number of base pattern alphabets. Figure 3.28(b) depicts the number of pattern alphabets and the corresponding number of patterns under its equivalence class that are not individual activities (alphabet size > 1) and having a conservedness value of at least 50%. It is to be noted that although the number of patterns uncovered in the event log are overwhelming, the number of significant patterns is just a small fraction (less than 5%) of the total number. One can exploit this factor and further reduce the time for computing the count metrics if all the three metrics, viz., OAC, NOAC, and NOGAC are to be computed. One can first estimate only the overlapping counts for all the patterns and filter those that are not

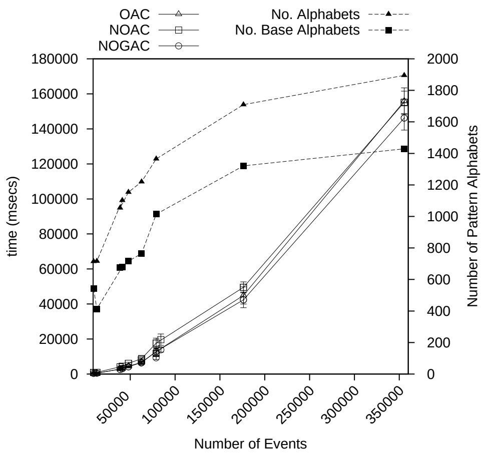


(a) suffix tree construction

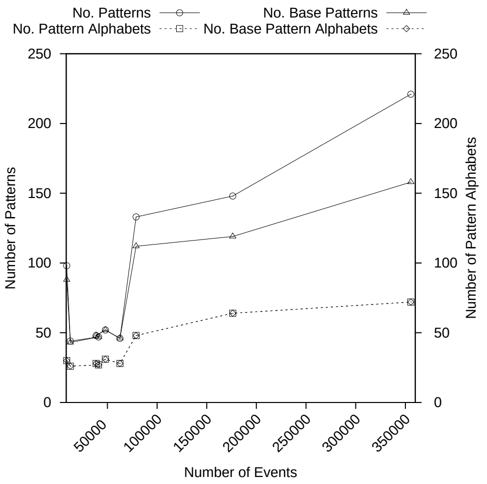


(b) discover all maximal repeats

Figure 3.27: Computational time to construct the suffix tree for the combined sequence obtained by concatenating all traces and to discover all maximal repeats for varying sizes of the event log. The 95% confidence intervals are also indicated.



(a) time to compute various count metrics



(b) number of significant patterns (conservedness value above 50% and alphabet size > 1)

Figure 3.28: Computational time required to compute various count metrics and the number of significant patterns (and pattern alphabets). The 95% confidence intervals are also indicated.

significant. The NOAC and NOGAC metrics can then be computed for the reduced set of the patterns thereby saving time. Figure 3.29 depicts the time taken to compute the NOAC and NOGAC metrics when the patterns are filtered based on their instance percentage count and conservedness value obtained after the estimation of the overlapping alphabet counts.

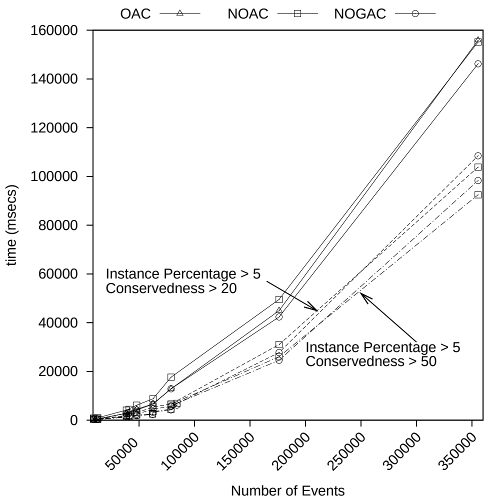
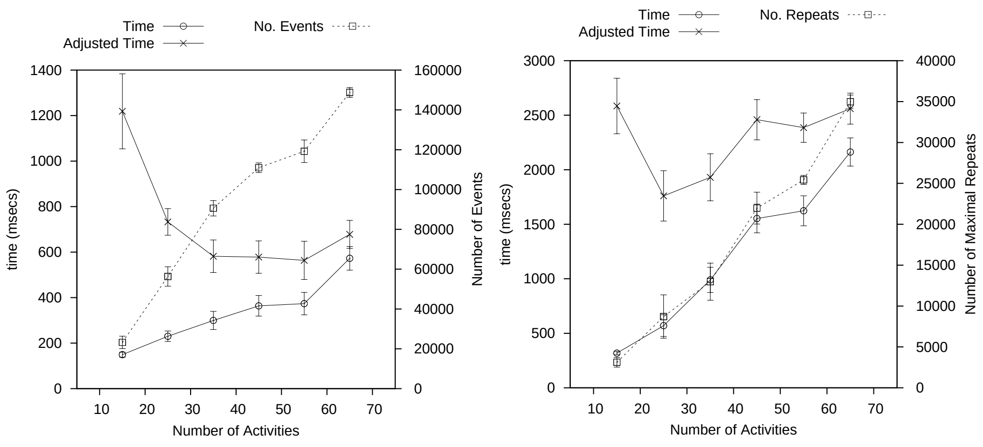


Figure 3.29: Computational time required to compute the various count metrics. The computational time can be reduced by apriori filtering of insignificant patterns.

3.6.2 Influence of Alphabet Size

In order to study the influence of alphabet size on pattern discovery, we considered a log $\mathcal{L}$ with 500 cases and 176,153 events referring to 77 activities. We randomly selected alphabets of varying sizes (15, 25, 35, 45, 55, and 65) from this log and filtered the rest. For each alphabet size, we generated five different logs each corresponding to a different random selection of activities. The patterns are uncovered for each such filtered log and the experiment is repeated five times. In other words, for each alphabet size, we have data generated from 25 runs. Figure 3.30(a) depicts the average time taken along with the 95% confidence intervals to construct the suffix tree. Recall that the construction of suffix tree of a sequence varies linearly with the size of the sequence and this is reflected in Figure 3.30(a). The size of the event log is bound to change with the alphabet size and hence the trend observed in Figure 3.30(a) is a result of the dual effect of both the alphabet size and the size of the event log.

In order to investigate the influence of just the alphabet size, we consider the adjusted time that is an extrapolation of time for a fixed size log. The adjusted time for a run is computed by multiplying the time to construct the suffix tree in that run by a scale factor, $|\mathcal{L}|/|\mathcal{L}'|$ where $|\mathcal{L}|$ is the size of the event log (176,163) and $|\mathcal{L}'|$ is the size of the filtered log considered in that run. We can see that the alphabet size does not have a significant influence on the suffix tree construction time (the adjusted time for the alphabet sizes of 15 and 25 are outliers owing to a high scaling factor). Figure 3.30(b) depicts the average time along with the 95% confidence intervals to uncover the maximal repeats over logs with varying alphabet size. Recall that the discovery of maximal repeats varies linearly with the number of repeats and this



(a) suffix tree construction (b) discovering maximal repeats

Figure 3.30: Computational time required to construct the suffix tree and uncover maximal repeats over logs with varying alphabet size. The 95% confidence intervals are also indicated.

is clearly reflected in Figure 3.30(b). In order to isolate the influence of just the alphabet size, we consider the adjusted time as above. The influence of alphabet size on pattern discovery is on expected lines in that it is proportional to the differences in the repeats induced by the increase in activities.

3.7 Limitations and Extensions

In the previous section, we have *seen the promise* of the proposed approach in forming meaningful abstractions at a manageable computational complexity even for extremely large logs. Nonetheless, the approach suffers from a few limitations. In this section, we highlight the limitations and propose extensions to address some of these limitations.

3.7.1 Demarcation of Functional Boundaries

The pattern definitions considered in this chapter lack the ability to clearly demarcate boundaries between functionalities. In other words, a pattern $\alpha = \beta_1 \diamond \beta_2$ might contain activity subsequences β_1 and β_2 belonging to two different subprocesses that are executed one after the other. This is due to the fact that a subsequence constitutes a maximal repeat only if it has distinct contexts. This can be partially mitigated by semi-automated means where an analyst can split a pattern or remove activities from a pattern, e.g., in the above pattern, an analyst can split α into two patterns β_1 and β_2. This feature is supported in the Pattern Abstractions plug-in (cf. Chapter 10).

3.7.2 Robust Pattern Metrics and Pruning Techniques

The conservedness metric proposed in this chapter is biased towards base patterns, i.e., patterns where no activity repeats itself. A high conservedness value is assigned to those patterns whose frequency is close to the mean frequency of the individual activities defined by it. In cases where a pattern has some activities repeating within it, the count of the pattern can be much lower than the mean. Hence, there is a need for additional robust metrics to assess the significance of patterns.

There is also a need for additional pruning techniques to extract all significant patterns. There could be scenarios where a significant pattern is not explicitly captured as a pattern but subsumed within a larger pattern. For example, consider a pattern $\alpha = \mathtt{x} \diamond \beta \diamond \mathtt{y}$ where the activities $\mathtt{x}$ and $\mathtt{y}$ do not occur in the subsequence β. Let us also assume that the subsequence β always occurs in the context of $\mathtt{x}$ and $\mathtt{y}$ but the activities $\mathtt{x}$ and $\mathtt{y}$ can also occur in different contexts. The subsequence β cannot be captured as a maximal repeat because its context to the left and right is always $\mathtt{x}$ and $\mathtt{y}$. However, the conservedness value of the pattern α can be low because the frequency of occurrence of $\mathtt{x}$ and $\mathtt{y}$ is different from that of β. One can tackle such patterns by pruning the activities at the boundaries provided there is an improvement in the conservedness value.

3.7.3 Patterns in the Manifestation of Complex Process Model Constructs

The pattern definitions defined in Section 3.3 (both tandem arrays, maximal repeats and its variants) capture some important manifestations of the process model constructs, but they are not sufficient enough to cater to complex model constructs, e.g., a parallelism or choice within other constructs such as loops. We call the above pattern definitions to be *exact*. The *merging* of maximal elements in the Hasse diagram addresses this to a certain extent. In order to deal with complex constructs, the pattern definitions need to be more flexible and robust. In this section, we address some of these pattern definitions and call these *approximate*.

Definition 3.21 (Approximate Tandem Arrays). An *approximate tandem array* in a sequence $\mathbf{t}$ is a concatenation of sequences $\alpha = \mathbf{s}_1 \diamond \mathbf{s}_2 \diamond \mathbf{s}_3 \diamond \cdots \diamond \mathbf{s}_k$ for which there exists a sequence $\mathbf{s}_c$ such that each $\mathbf{s}_i$ $(1 \leq i \leq k)$ is *approximately similar* to $\mathbf{s}_c$. Here, $\mathbf{s}_c$ can be different from each and every $\mathbf{s}_i$; alternatively, we may constrain that $\mathbf{s}_c$ be equal to at least one $\mathbf{s}_i$ $(1 \leq i \leq k)$. The sequence $\mathbf{s}_c$ is called as the *primitive approximate tandem repeat type*, and the approximate tandem array α is represented by the triple $(j, \mathbf{s}_c, k)$ where j signifies the starting position of α in $\mathbf{t}$. ⌟

The notion of similarity can be defined in multiple ways (such as the Hamming distance and string edit distance). For example, two sequences with a string edit distance [184] less than δ (for some threshold, $\delta \in \mathbb{N}$) can be considered to be similar.

Approximate tandem arrays can be used to detect choices within loops. For example, consider the process model construct depicted in Figure 3.31(a). In this example, we have a choice construct over the activities `b` and `c` inside the loop. `s` = `abdacdacdabd` is one manifestation of the construct that constitutes an approximate tandem array with `abd` or `acd` as an approximate primitive tandem repeat type. Here, the Levenshtein string edit distance [136] with a threshold of 2 is used as the notion of approximation. Parallelism within loops can also be handled in a similar fashion by approximate tandem arrays. However, defining an appropriate notion of similarity is crucial for the success of this approach. A too lenient notion might generate too many false positives while a stringent notion will miss certain constructs. This problem is compounded by the number of activities involved in the parallelism construct.

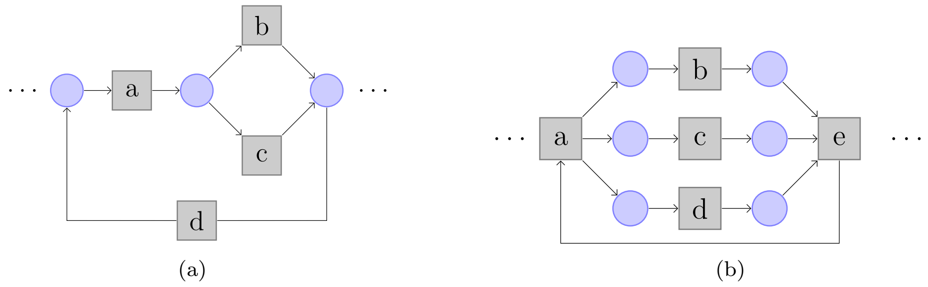


Figure 3.31: Complex process model constructs (a) choice within loops (b) parallelism within loops.

Just like the approximate tandem arrays, we can define notions of approximation for non-tandem repeats (maximal repeats, super-maximal, and near-super maximal repeats). Approximate repetitions are specified by allowing some number of "errors" between repeated copies. The set of allowed errors can be defined under the Hamming distance and the edit distance framework. If replacements alone are allowed, this yields the classic Hamming distance, defined as the number of mismatches between the two sequences; if both replacements and insertions/deletions are permitted, then we are operating in the edit distance framework.

Definition 3.22 (Approximate Repeat). A repeat pair (α, α') is a k−approximate repeat if and only if the distance between them $d(\alpha, \alpha') \le k$ (for some $k \in \mathbb{N}$). ⌟

Consider Figure 3.31(b) that contains a parallelism construct. $\mathtt{s}_1$ = `abcde`, $\mathtt{s}_2$ = `acdbe`, $\mathtt{s}_3$ = `adcbe` are some of the manifestations of the construct. The pairs (`abcde`, `acdbe`), (`acdbe`, `adcbe`), and (`abcde`, `adcbe`) are all 2−approximate under the Levenshtein edit distance.

Sokol et al. [205] proposed an approach for a variant of approximate tandem arrays under the edit distance with $\mathcal{O}(nk \log k \log(n/k))$ time and $\mathcal{O}(n + k^2)$ space complexity (where n is the length of the sequence and k is the threshold on the edit distance for similarity).

3.8 Conclusions

Current approaches to process mining suffer from the gap between the granularity of events recorded by an information system and an analysts' perspective of a business process. In this chapter, we made an attempt at paving a way to bridge this gap. We exploited the common execution patterns manifested in an event log, correlated them to commonly used process model constructs, and proposed a means of forming abstractions of activities. The patterns can be efficiently discovered in linear time and space. Hence, they can be applied to challenging real-life event logs. The abstractions thus formed can be used to simplify the log and lift them to a desired level of granularity (low-level events in the log are replaced with abstract activities). We will look at this transformation in more detail in Chapter 6.

Chapter 4
Trace Clustering

In the previous chapter, we looked at techniques for dealing with fine-granular event logs. In this chapter, we look at techniques to deal with the *heterogeneity* in event logs. Process mining techniques have problems dealing with variability, which is caused by heterogeneity, i.e., different usage scenarios are merged into a single *spaghetti-like* process. In such scenarios, multiple comprehensible models capturing different classes of behavior are preferred over a single spaghetti-like model. One means of achieving this is to preprocess an event log to segregate/cluster homogenous sets of cases and analyzing each set of homogenous cases separately. Figure 4.1 illustrates the significance of this approach. The process model on the top of Figure 4.1 is a process model mined from a heterogenous event log. This model is quite complex to comprehend. The bottom rectangles of Figure 4.1 depict the process models mined from the clustered traces. It is evident that clustering enables the comprehension of process models by reducing the spaghettiness. Such a segregation also assumes importance in dealing with voluminous data, going by the adage of "divide and conquer". Furthermore, event logs may contain anomalous, outlier and/or noisy traces. The presence of such impurity in logs can also impact the goodness of the mined results.

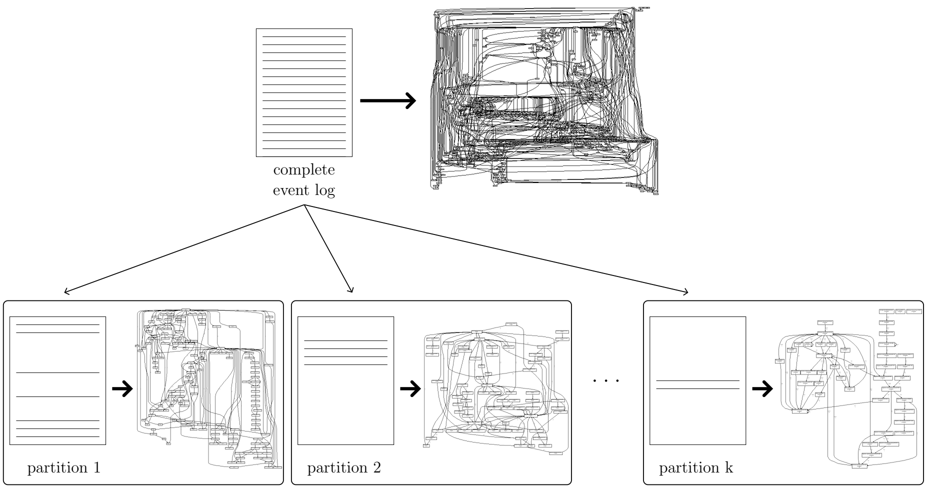


Figure 4.1: Significance of trace clustering in process mining.

Clustering is an unsupervised data mining technique focusing on grouping together of homogenous objects [104]. The objective of clustering is to retrieve groups, or clusters, of data objects such that the objects within a cluster are as similar as possible to each other while the objects belonging to two different clusters are dissimilar, as illustrated in Figure 4.2. In process mining, the objects are the process instances (i.e., cases) in an event log. It is imperative that a notion of (dis-)similarity between process instances needs to be defined to enable the formation of clusters.

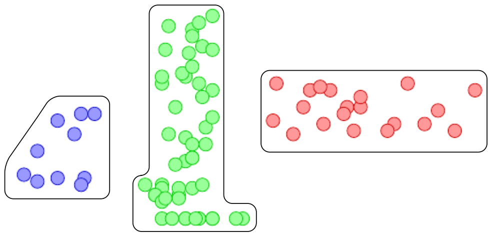


Figure 4.2: An example of segregating objects into three clusters.

There are two primary approaches to defining the (dis-)similarity between cases. One approach, called the *vector-based approach*, defines a feature space (e.g., activities, data attributes, transitions, resources, etc.) and transforms each case into a vector in the feature space. Once cases are represented as vectors, several measures to calculate the distance/similarity[1] between them can be used and the choice of the measures largely depends on the type of features composing the data [104]. Another approach is the *syntactic approach*, where the cases are viewed as *traces*, i.e., sequences composed of symbols belonging to an alphabet (where the alphabet corresponds to some character encoding of the activities or resources etc.). One can then use edit distance measures [184] to define the (dis-)similarity between sequences and, thereby, between traces.

In this chapter, we emphasize the significance of considering the contexts of execution in clustering. In the vector based approach, such contexts need to be captured in the feature sets while the edit distance measures do that through a cost/score function for the edit operations. We propose a few feature sets based on common execution patterns (defined in Chapter 3) in an event log. The common execution patterns in addition to being context-aware also reflect the manifestation of some process model constructs thereby easing the interpretation of the resulting clusters. For the edit distance measure, we propose an automated approach to derive the scores for the edit operations. We also propose objective metrics to compare and assess the goodness of the formed clusters from a process mining point of view.

The remainder of this chapter is organized as follows. Section 4.1 presents the

[1]Distance measures are often used to capture the notion of dissimilarity.

related work on clustering event logs in process mining applications. Section 4.2 presents well-known and often used feature sets and their pitfalls, and proposes context-aware process-centric feature sets. Section 4.3 discusses the syntactic approach to clustering traces. Section 4.4 presents an automated approach to derive the scores for the edit operations. Section 4.5 presents an overview of the hierarchical clustering algorithm while Section 4.6 discusses the computational complexity of trace clustering. Section 4.7 presents a few evaluation metrics to assess the goodness of clusters from a process mining point of view. The experimental results and scalability issues are discussed in Section 4.8. Section 4.9 presents some extensions for the proposed techniques. Finally, Section 4.10 concludes the chapter.

4.1 Related Work

Data clustering is one of the most important topics in data mining and many algorithms exist in the literature [104, 113]. The significance of trace clustering to process mining has been first discussed in [54, 87]. Greco et al. [87] capture structural properties (parallelism and precedence relations between activities) manifested in an event log as constraints and use them as features. The traces are projected onto these features and transformed into a vector space. Standard clustering algorithms can then be applied to cluster the traces. Alves de Medeiros et al. [54] use a similar approach to cluster an event log iteratively until each of the resulting clusters can be adequately represented by a process model. Greco et al. [88] extend their work on using structural information to also consider performance measures such as the duration of activity executions in traces and propose an information theoretic framework based on co-clustering techniques. More recently, Greco et al. [72, 80] have extended the work on using structural patterns by proposing new patterns, viz., FORK and JOIN that also consider the effects of interleaving of parallel branches and showed its applicability in detecting outlier traces. In contrast to transforming the traces into a vector space, they use co-clustering techniques [59] by looking at associations between traces and patterns. Song et al. [208, 209] have proposed the idea of clustering traces by considering a combination of different perspectives of the traces (such as activities, transitions, data, performance, etc.) to define the feature space. From a control-flow perspective, Song et al. [209] use the bag-of-activities and transitions between activities as the feature sets. Transitions can be considered as a special case of k-grams (sequences of k activities) where the value of k is 2. We discuss the pitfalls of considering the bag-of-activities and k-grams later in this chapter.

The feature sets proposed in this chapter capture constraints between activities by exploiting common execution patterns that are related to the manifestation of some process model constructs such as loops and choice constructs. Furthermore, variations resulting from parallelism are captured under equivalence classes of pattern alphabets. Other perspectives of the traces such as data, organization and performance as proposed in [208, 209] can be seamlessly integrated and used complementarily to the feature sets proposed in this chapter. Moreover, the proposed feature sets can be efficiently discovered in linear time making it amenable to large scale event logs.

Diogo et al. [69, 70, 249] use a sequence clustering algorithm [30] to partition an event log. The adopted approach is a model-based clustering technique that relies on the Expectation-Maximization procedure [56]. The basic idea is to represent a cluster by a Markov chain and assign each input sequence to that cluster that is most likely to produce the sequence. The cluster model is iteratively updated until convergence from the set of sequences assigned to it. This approach is sensitive to the initialization of the cluster models and is computationally expensive. In Section 4.8, this is discussed in more detail.

4.2 Vector Based Approaches to Trace Clustering

As mentioned earlier, in the vector-based approach to trace clustering, one needs to define a feature space and transform each trace in the event log into a vector in this feature space. During this transformation, one can either use a *binary representation* or a *numeric representation* for a trace. In the binary representation, we consider only the presence or absence of a feature in a trace and represent the value for that feature as 1 or 0 respectively, whereas in the numeric representation, the value for a feature corresponds to the frequency of occurrence of that feature in a trace.

4.2.1 A Critical Analysis of Contemporary Feature Sets

Several approaches have been proposed in the literature to transform the cases into a vector-space [208, 209] pertaining to the different perspectives of process mining, viz., the control-flow, data, organization, and time perspectives. In this section, we consider only the feature sets defined on the control-flow perspective.

Bag-of-activities (BOA)

One of the most often used feature sets for trace clustering is the bag-of-activities. This feature set corresponds to the set of all activities (or tasks) present in the event log. For example, the traces `abaac` and `badca` correspond to the vectors (3, 1, 1, 0) and (2, 1, 1, 1) respectively in the numeric representation and to (1, 1, 1, 0) and (1, 1, 1, 1) in the binary representation; the dimensions of the vector being (`a`, `b`, `c`, `d`). The numeric representation of the traces using this feature set corresponds to the *Parikh vectors*. The bag-of-activities feature set has a few drawbacks:

- *Lack of context information:* Process execution is characterized by contexts. The bag-of-activities representation does not capture the dynamics of process execution. As an example, consider a process model with a *notification* activity. The different instances of notification within a trace might have different connotations based on the context in which it is invoked. For example, a broadcast notification, notification requesting a response, etc.
- *Order of execution:* The bag-of-activities representation also looses the information on the order of execution of events. Any permutation of the sequence of

activities of a given trace has the same vector representation and thereby has a distance of 0 with each other, thus being less robust to noise and exceptional behavior. However, in reality, a lot of these permutations do not make any sense from a process definition point of view. For example, in the digital copier example, we cannot have a `Send by email` activity (for scanned documents) before the `Image Processing` is done.

One means of incorporating context is to consider subsequences of activities. These subsequences capture the order of execution as well. However, it is important to note that the notion of context can be much more than a mere order of execution of activities.

k-grams

Another commonly used feature set is the k-gram. A k-gram refers to a subsequence of k activities. For the trace `abacaab`, the set of 2-grams (also called bi-grams) corresponds to {`ab`, `ba`, `ac`, `ca`, `aa`} while the set of 3-grams (tri-grams) corresponds to {`aba`, `bac`, `aca`, `caa`, `aab`}. It is to be noted that the size of this feature set increases drastically with an increase in the size of the alphabet $|\mathcal{A}|$ and/or k. For example, considering 3-grams in an event log with 100 activities can potentially lead to 10^6 features. Although in reality one may not see all combinations of 3-grams in the event log, working in the k-gram space incurs a huge computational overhead. In addition, selecting a suitable value for k is non-trivial.

Because of the problems just mentioned, we propose alternative feature sets tailored towards process mining.

4.2.2 Process-Centric Feature Sets

In this section, we propose new feature sets for the vector-space model that are context-aware. The basic idea is to explore whether traces in an event log can be grouped based on recurring patterns (common subsequences of activities). The premise is that the distribution of the patterns indicate similarity between the traces that incorporate them. Unlike the k-gram approach where we consider subsequences of k-activities (for a fixed k), these feature sets are based on subsequences of different lengths. In the previous chapter, we have presented some sequence patterns and established their relationship to some process model constructs, e.g., tandem arrays capture the manifestation of loop constructs. We exploit this correspondence and define multiple feature sets. The feature sets can be classified into two categories: (i) *sequence features*, and (ii) *alphabet features*. Sequence features enforce a strict constraint on the order of activities while alphabet features, derived from sequence features, relax the ordering within similar sequence features, e.g., two common subsequences of activities `abc` and `bac` would be considered different in sequence features whereas they would be grouped together into one under the alphabet features. Given an event log $\mathcal{L}$, we define the following feature sets:

- *Tandem Repeats* (TR): The features correspond to the primitive tandem repeat types in the event log $\mathcal{L}$. Those primitive tandem repeat types that are individual activities are removed. Including repeats that are individual activities lead to the drawbacks mentioned for the bag of activities approach.

$$\text{TR} = \{\mathbf{r} \mid \mathbf{r} \text{ is a primitive tandem repeat type in } \mathcal{L} \text{ and } |\mathbf{r}| > 1\}$$

- *Maximal Repeats* (MR): The features correspond to the maximal repeats in the event log $\mathcal{L}$.

$$\text{MR} = \{\mathbf{r} \mid \mathbf{r} \text{ is a maximal repeat in } \mathcal{L} \text{ and } |\mathbf{r}| > 1\}$$

- *Near Super Maximal Repeats* (NSMR): This feature set is based on the *near super maximal repeats* in the event log $\mathcal{L}$ and is defined as

$$\text{NSMR} = \{\mathbf{r} \mid \mathbf{r} \text{ is a near super maximal repeat in } \mathcal{L} \text{ and } |\mathbf{r}| > 1\}$$

- *Super Maximal Repeats* (SMR): This feature set is based on the *super maximal repeats* in the event log $\mathcal{L}$ and is defined as

$$\text{NSMR} = \{\mathbf{r} \mid \mathbf{r} \text{ is a super maximal repeat in } \mathcal{L} \text{ and } |\mathbf{r}| > 1\}$$

- *Tandem Repeat Alphabet* (TRA): This is a derived feature set of TR and is defined as

$$\text{TRA} = \{\Gamma(\mathbf{r}) \mid \mathbf{r} \in \text{TR and } |\Gamma(\mathbf{r})| > 1\}^2$$

- *Maximal Repeat Alphabet* (MRA): This is a derived feature set of MR and is defined as

$$\text{MRA} = \{\Gamma(\mathbf{r}) \mid \mathbf{r} \in \text{MR and } |\Gamma(\mathbf{r})| > 1\}$$

- *Near Super Maximal Repeat Alphabet* (NSMRA): This is a derived feature set of NSMR and is defined as

$$\text{NSMRA} = \{\Gamma(\mathbf{r}) \mid \mathbf{r} \in \text{NSMR and } |\Gamma(\mathbf{r})| > 1\}$$

- *Super Maximal Repeat Alphabet* (SMRA): This is a derived feature set of SMR and is defined as

$$\text{SMRA} = \{\Gamma(\mathbf{r}) \mid \mathbf{r} \in \text{SMR and } |\Gamma(\mathbf{r})| > 1\}$$

As discussed in Chapter 3, the alphabet variants capture variations in the manifestation of activities arising due to parallelism. There is no thumb rule or guideline on the appropriateness of a feature set and the choice of a feature set is largely dependent on the context of analysis. For example, if we want to cluster a log based on whether a particular loop construct is invoked or not, then choosing either TR or TRA as the feature set is recommended. The above feature sets are not orthogonal and are closely related, e.g., SMR $\subseteq$ NSMR $\subseteq$ MR and SMRA $\subseteq$ NSMRA $\subseteq$ MRA.

Consider the event log $\mathcal{L}$ = [`jgcflebd`, `jgclebdfkahbd`, `jgcahbd`, `sampcudn`, `samcupdn`]. The maximal repeat feature set in the event log corresponds to MR = {`bd`, `cu`, `dn`, `jgc`, `sam`, `ahbd`, `lebd`}. The traces in the event log can be transformed into vectors

$\mathcal{L}$	vector representation
$\mathbf{t}_1$ = `jgcflebd`	$(1, 0, 0, 1, 0, 0, 1)$
$\mathbf{t}_2$ = `jgclebdfkahbd`	$(2, 0, 0, 1, 0, 1, 1)$
$\mathbf{t}_3$ = `jgcahbd`	$(1, 0, 0, 1, 0, 1, 0)$
$\mathbf{t}_4$= `sampcudn`	$(0, 1, 1, 0, 1, 0, 0)$
$\mathbf{t}_5$= `samcupdn`	$(0, 1, 1, 0, 1, 0, 0)$

Euclidean distance

$$\begin{array}{c|ccccc|}
 & [\mathbf{t}_1] & [\mathbf{t}_2] & [\mathbf{t}_3] & [\mathbf{t}_4] & [\mathbf{t}_5] \\
\hline
[\mathbf{t}_1] & 0 & 1.41 & 1.41 & 2.45 & 2.45 \\
[\mathbf{t}_2] & & 0 & 1.41 & 3.16 & 3.16 \\
[\mathbf{t}_3] & & & 0 & 2.45 & 2.45 \\
[\mathbf{t}_4] & & & & 0 & 0 \\
[\mathbf{t}_5] & & & & & 0 \\
\hline
\end{array}$$

Figure 4.3: Transformation of traces into vector space and computation of the Euclidean distance between them. The dimensions of the vector correspond to (`bd`, `cu`, `dn`, `jgc`, `sam`, `ahbd`, `lebd`).

where the dimensions are defined by the maximal repeats. Figure 4.3 depicts the transformation of the traces in the event log $\mathcal{L}$ using the MR feature set and the computation of the Euclidean distance between the traces.

Having discussed the vector-based approaches to trace clustering, we next look at the syntactic approaches.

4.3 Syntactic Approaches to Trace Clustering

A trace in an event log corresponds to the sequence of activities executed in a process instance. While the vector-space model falls under the statistical processing domain, syntactic approaches to clustering consider the sequences as is and define the distance between sequences in terms of error transformations [73, 74]. The advantage of using syntactic methods is that it considers a trace in totality thereby preserving the context and ordering. Before presenting our approach, we first discuss standard measures for defining the distance based on error transformations.

Levenshtein Distance

A fundamental measure of (dis-)similarity between two sequences is the Levenshtein distance, also called as the edit distance. Levenshtein distance between two sequences is defined as the minimum number of edit operations needed to transform one sequence into the other, where an edit operation is an insertion, deletion, or substitution of an element [136]. Consider an alphabet $\mathcal{A}$ and two sequences $\mathbf{s}$ and $\mathbf{t} \in \mathcal{A}^+$. $\mathbf{s}$ and $\mathbf{t}$ may contain (i) symbols common to both of them, (ii) symbols present only in $\mathbf{s}$, and (iii) symbols present only in $\mathbf{t}$. For example, consider the two sequences $\mathbf{s}$ = `abcac` and $\mathbf{t}$ = `acacad`, $|\mathbf{s}| = 5$ and $|\mathbf{t}| = 6$. $\mathbf{s}$ and $\mathbf{t}$ have the symbols `a` and `c` in common while the symbol `b` occurs only in $\mathbf{s}$ and the symbol `d` occurs only in $\mathbf{t}$. There are many possibilities in which one can transform $\mathbf{s}$ into $\mathbf{t}$. One can delete symbols that occur only in $\mathbf{s}$ and insert symbols that occur only in $\mathbf{t}$ or one can replace certain

[2]Recall from Chapter 3 that $\Gamma(\mathbf{p})$ correponds to the pattern alphabet of the pattern $\mathbf{p}$.

symbols in $\mathbf{s}$ with symbols in $\mathbf{t}$. During the transformation of $\mathbf{s}$ to $\mathbf{t}$, for any $1 \leq i \leq |\mathbf{s}|$ and $1 \leq j \leq |\mathbf{t}|$, we can use the following three operators:

- $(\mathbf{s}(i), \mathbf{t}(j))$ denotes the replacement/substitution of the element $\mathbf{s}$(i) with $\mathbf{t}$(j).
- $(\mathbf{s}(i), -)$ denotes the deletion of the element $\mathbf{s}$(i).
- $(-, \mathbf{t}(j))$ denotes the insertion of the element $\mathbf{t}$(j).

The sequence $\mathbf{s}$ = `abcac` can be transformed into $\mathbf{t}$ = `acacad` through the edit operations defined by $(\mathbf{s}(2), -)$, $(-, \mathbf{t}(5))$, $(-, \mathbf{t}(6))$, i.e., by deleting the second symbol in $\mathbf{s}$ and inserting the last two symbols of $\mathbf{t}$ at the end of $\mathbf{s}$. The transformation between the sequences can be visualized as an alignment as depicted in Figure 4.4 ($\overline{\mathbf{s}}$ and $\overline{\mathbf{t}}$ correspond to the transformed sequences of $\mathbf{s}$ and $\mathbf{t}$ defined over the alphabet $\mathcal{A} \cup \{-\}$).

$$\begin{array}{lccccccc} \overline{\mathbf{s}} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - & - \\ \overline{\mathbf{t}} & \mathtt{a} & - & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d} \end{array}$$

Figure 4.4: Transformation of sequences visualized as an alignment.

It is important to note that the transformation of one sequence to another can be done in many different ways, e.g., Figure 4.5 depicts three of the many ways of transforming $\mathbf{s}$ to $\mathbf{t}$. In Figure 4.5(b), $\mathbf{s}$ is transformed into $\mathbf{t}$ using four substitution operations[3], $(\mathtt{b}, \mathtt{c}), (\mathtt{c}, \mathtt{a}), (\mathtt{a}, \mathtt{c}), (\mathtt{c}, \mathtt{a})$ and an insertion operation, $(-, \mathtt{d})$. *Levenshtein distance corresponds to the minimum number of edit operations required to transform one sequence into the other.* The Levenshtein distance between the above two traces is 3.

$$\begin{array}{lccccccc} \overline{\mathbf{s}} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - & - \\ \overline{\mathbf{t}} & \mathtt{a} & - & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d} \end{array}$$

(a)

$$\begin{array}{lcccccc} \overline{\mathbf{s}} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - \\ \overline{\mathbf{t}} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d} \end{array}$$

(b)

$$\begin{array}{lccccccccccc} \overline{\mathbf{s}} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - & - & - & - & - & - \\ \overline{\mathbf{t}} & - & - & - & - & - & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d} \end{array}$$

(c)

Figure 4.5: Three of the many ways of transforming $\mathbf{s}$ = `abcac` to $\mathbf{t}$ = `acacad`.

Levenshtein distance, though noteworthy for its simplicity, does not fit in many application scenarios. Consider the digital copier example and the following scenarios of copy/scan jobs:

- for the event traces $\mathbf{t}_1$ = ⟨`Heat Roller Spin Start`, `Apply Heat`, `Heat Roller Spin Stop`, `Pressure Roller Spin Start`, `Apply Pressure`, `Pressure Roller Spin Stop`⟩ and $\mathbf{t}_2$ = ⟨`Pressure Roller Spin Start`, `Heat Roller Spin Start`, `Apply Pressure`, `Apply Heat`, `Pressure Roller Spin Stop`, `Heat Roller Spin Stop`⟩, the Levenshtein distance between $\mathbf{t}_1$ and $\mathbf{t}_2$ is 6 corresponding to the transformation depicted in Figure 4.6. However, it is to be

[3]Note that the error transformation considers only the substitution or replacement of unlike symbols, i.e., (x, y) with x ≠ y.

noted that, from an application point of view, the sequence of activities in the traces $\mathbf{t}_1$ and $\mathbf{t}_2$ are similar in that both correspond to the Fusing subprocess where the application of pressure and heat are in parallel branches (cf. Figure 3.9).

- now, consider another trace $\mathbf{t}_3$ = ⟨`Writing`, `Developing`, `Fusing`, `Wipe Toner on Drum`, `Erase Toner on Drum`⟩ corresponding to the subprocess, Print Image (cf. Figure 3.6). The Levenshtein distance between $\mathbf{t}_1$ and $\mathbf{t}_3$ is 6, the transformation of which is depicted in Figure 4.7; so is the Levenshtein distance between $\mathbf{t}_2$ and $\mathbf{t}_3$. It is obvious that $\mathbf{t}_1$ is more similar to $\mathbf{t}_2$ (both belong to the Fusing subprocess) than $\mathbf{t}_3$ is, which belongs to the Print Image subprocess. However, this is blurred by the equal Levenshtein distance measures.

———	Heat Roller Spin Start	———	Apply Heat	———	Heat Roller Spin Stop	Pressure Roller Spin Start	Apply Pressure	Pressure Roller Spin Stop
Pressure Roller Spin Start	Heat Roller Spin Start	Apply Pressure	Apply Heat	Pressure Roller Spin Stop	Heat Roller Spin Stop	———	———	———

Figure 4.6: Transformation with a Levenshtein distance of 6 between the traces $\mathbf{t}_1$ and $\mathbf{t}_2$.

Heat Roller Spin Start	Apply Heat	Heat Roller Spin Stop	Pressure Roller Spin Start	Apply Pressure	Pressure Roller Spin Stop
Writing	Developing	Fusing	Wipe Toner on Drum	Erase Toner on Drum	———

Figure 4.7: Transformation with a Levenshtein distance of 6 between the traces $\mathbf{t}_1$ and $\mathbf{t}_3$.

In other words, the Levenshtein distance does not consider the functional validity of any edit operation. Also, two sequences of lengths l_1 and l_2, irrespective of their similarity, will always have a Levenshtein distance of at least $|l_1 - l_2|$, where $|l|$ denotes the absolute value of l. It is quite natural for event log traces to be of different lengths. For example, consider the two traces $\mathbf{t}_4$ = `abacd` and $\mathbf{t}_5$ = `abacacacacd`. These two traces are similar in that they would have been generated from the same process model where there is a loop construct over the activities `ac`. Ideally, one would like to put these two traces in the same cluster. If we apply the Levenshtein metric, we get a distance of 6. Now consider another trace $\mathbf{t}_6$ = `jgcle`, which corresponds to a totally different functionality. The Levenshtein distance between $\mathbf{t}_4$ and $\mathbf{t}_6$ is 5, which is lesser than that of $\mathbf{t}_4$ and $\mathbf{t}_5$. This poses a danger of preferring $\mathbf{t}_6$ over $\mathbf{t}_5$ to be clustered together with $\mathbf{t}_4$. Therefore, one should consider the manifestations of process model constructs to alleviate such problems.

One means of alleviating the problems with the Levenshtein distance is to consider costs or weights for the edit operations. The generic edit distance framework captures such cost structures.

Generic Edit Distance

The generic string edit distance is characterized by a triple $\langle \mathcal{A}, \mathcal{B}, \mathcal{C} \rangle$ consisting of finite alphabets $\mathcal{A}$ and $\mathcal{B}$ and the primitive cost function $\mathcal{C} : E \to \mathbb{R}_0^+$ where $E = E_s \cup E_i \cup E_d$ is the set of primitive edit operations on the alphabets. $E_s = \mathcal{A} \times \mathcal{B}$ is the set of substitutions, $E_d = \mathcal{A} \times \{-\}$ is the set of deletions, and $E_i = \{-\} \times \mathcal{B}$ is the set of insertions. In many applications $\mathcal{A} = \mathcal{B}$. The generic edit distance over a bag of sequences $\mathbb{T}$ is a function, $ged : \mathbb{T} \times \mathbb{T} \to \mathbb{R}_0^+$. The distance between two sequences $\mathbf{s}$ and $\mathbf{t}$ is defined under the generic edit distance as

$$ged(\mathbf{s}, \mathbf{t}) = \min \begin{cases} \mathcal{C}(\mathbf{s}(|\mathbf{s}|), \mathbf{t}(|\mathbf{t}|)) + ged(\mathbf{s}^{|\mathbf{s}|-1}, \mathbf{t}^{|\mathbf{t}|-1}), \\ \mathcal{C}(\mathbf{s}(|\mathbf{s}|), -) + ged(\mathbf{s}^{|\mathbf{s}|-1}, \mathbf{t}), \\ \mathcal{C}(-, \mathbf{t}(|\mathbf{t}|)) + ged(\mathbf{s}, \mathbf{t}^{|\mathbf{t}|-1}) \end{cases} \tag{4.1}$$

When either $\mathbf{s} = \langle\rangle$ or $\mathbf{t} = \langle\rangle$, i.e., when either of the sequences is empty, only insertions or deletions are allowed. Thus,

$$\begin{aligned} ged(\mathbf{s}, \langle\rangle) &= \mathcal{C}(\mathbf{s}(|\mathbf{s}|), -) + ged(\mathbf{s}^{|\mathbf{s}|-1}, \langle\rangle), \\ ged(\langle\rangle, \mathbf{t}) &= \mathcal{C}(-, \mathbf{t}(|\mathbf{t}|)) + ged(\langle\rangle, \mathbf{t}^{|\mathbf{t}|-1}), \\ ged(\langle\rangle, \langle\rangle) &= 0 \end{aligned} \tag{4.2}$$

It is important to note that insertions and deletions are complementary in that an insertion in one trace can be considered as a deletion in another trace. Henceforth, we refer to insertion and deletion operations as an *indel* operation. Furthermore, unlike the error transformations [73, 74], the generic edit distance framework also considers the substitution of 'like' symbols as an edit operation. Edit distance can be efficiently computed using dynamic programming [150, 255]. The Levenshtein distance is a special case of the generic edit distance where a *unit cost* function is used for the edit operations. In other words, under Levenshtein distance, $\mathcal{C}(\mathtt{a}, \mathtt{a}) = 0, \mathcal{C}(\mathtt{a}, -) = \mathcal{C}(-, \mathtt{a}) = 1$, and $\mathcal{C}(\mathtt{a}, \mathtt{b}) = 1$ for $\mathtt{a} \neq \mathtt{b}$. In order to avoid edit operations that do not make sense in a certain context, the cost function, $\mathcal{C}$, needs to be more robust: substitution of uncorrelated/constrasting activities or insertion/deletion of activities not conforming to a context should be penalized heavily. On the other hand, 'related' events should be allowed to be replaced/inserted at a minimal cost. However, deriving such costs is nontrivial unless provided by a domain expert. In the next section, we propose an approach to derive the edit operation costs from event log traces and show that the derived costs have statistical as well as semantic significance.

4.4 Deriving Substitution and Indel Scores

Distance and similarity measures are interchangeable in the sense that a small distance means high similarity, and vice versa. For two sequences, $\mathbf{s}$ and $\mathbf{t}$, the edit distance in Equations (4.1) and (4.2) defined earlier can be transformed to a similarity function defined as

$$sim(\mathbf{s},\mathbf{t}) = \max \begin{cases} \mathcal{S}(\mathbf{s}(|\mathbf{s}|),\mathbf{t}(|\mathbf{t}|)) + sim(\mathbf{s}^{|\mathbf{s}|-1},\mathbf{t}^{|\mathbf{t}|-1}), \\ \mathcal{I}(\mathbf{s}(|\mathbf{s}|),-) + sim(\mathbf{s}^{|\mathbf{s}|-1},\mathbf{t}), \\ \mathcal{I}(-,\mathbf{t}(|\mathbf{t}|)) + sim(\mathbf{s},\mathbf{t}^{|\mathbf{t}|-1}) \end{cases} \tag{4.3}$$

$$\begin{aligned} sim(\mathbf{s},\langle\rangle) &= \mathcal{I}(\mathbf{s}(|\mathbf{s}|),-) + sim(\mathbf{s}^{|\mathbf{s}|-1},\langle\rangle), \\ sim(\langle\rangle,\mathbf{t}) &= \mathcal{I}(-,\mathbf{t}(|\mathbf{t}|)) + sim(\langle\rangle,\mathbf{t}^{|\mathbf{t}|-1}), \\ sim(\langle\rangle,\langle\rangle) &= 0 \end{aligned} \tag{4.4}$$

$\mathcal{S} : \mathcal{A} \times \mathcal{B} \to \mathbb{R}$ defines the substitution scores over the set of symbols and $\mathcal{I} : (\mathcal{A} \times \{-\}) \cup (\{-\} \times \mathcal{B}) \to \mathbb{R}$ defines the indel scores. In most applications $\mathcal{A} = \mathcal{B}$. Before we discuss the algorithm, we mention some of the desirable characteristics of substitution and indel scoring functions:

- substitution of uncorrelated activities should be discouraged
- substitution of contrasting activities (e.g., start and complete transaction types) should be penalized
- insertion of activities out of context should be discouraged
- substitution of correlated/similar activities should be encouraged in proportion to their degree of similarity

4.4.1 Substitution Scores

Algorithm 4.1, adapted from [8, 105, 121], depicts an approach to generate the substitution scores. The algorithm tries to maximize the score of two sequences based on similarity. In other words, it derives scores for substitution such that sequences that are similar attain a high score and sequences that are not similar get a low score. In contrast, the edit-distance assigns a lower value for similar sequences[4]. *The basic idea of the algorithm is to consider pairs of symbols that occur in similar contexts and compare their observed frequencies in the event log to their expected frequencies if occurred independently.* We assign high scores for substitution of those pairs of activities whose frequency of occurrence within certain contexts is more common than by chance.

Let us discuss the algorithm with an example. Consider the event log $\mathcal{L}$ = [aabcdbbcda, dabcdabcbb, bbbcdbbbccaa, aaadab, aacdbcbadbdebd] defined over the alphabet $\mathcal{A}$ = {a, b, c, d, e}. The set of 3-grams in $\mathcal{L}$ is G_3 = {aaa, aab, aac, aad, abc, acd, ada, adb, bad, bbb, bbc, bcb, bcc, bcd, bde, caa, cba, cbb, cca, ccc, cda, cdb, dab, dbb, dbc, dbd, deb, ebd}. The corresponding frequencies of the 3-grams are represented by the vector $\vec{F}_3 = (1,1,1,1,3,1,1,1,1,2,3,2,1,4,1,1,1,1,1,1,2,3,3,2,1,1,1,1)$ (Step 2, Algorithm 4.1). Let us define a *context* of a symbol a $\in \mathcal{A}$ to be the concatenation of the symbols to the immediate left and right of a manifested in the event log. In other words, the context of a symbol a is the subsequence x $\diamond$ y such

[4]We will later define a transformation between the similarity score and the distance value.

Algorithm 4.1 Algorithm to derive substitution scores

```
1: Let $\mathcal{A}$ be the alphabet; $\mathtt{x}, \mathtt{y}, \mathtt{a}, \mathtt{b} \in \mathcal{A}$
2: Let $G_3$ denote the set of all 3-grams present in the event log $\mathcal{L}$, and let $F_3 : G_3 \rightarrow \mathbb{N}$ denote their corresponding frequencies
3: Define $\mathcal{X}_{\mathtt{a}}$ to be the set of all contexts of the symbol $\mathtt{a}$. A context of a symbol $\mathtt{a}$ is the subsequence $\mathtt{xy}$ ($\mathtt{x} \diamond \mathtt{y}$) such that $\mathtt{x} \diamond \mathtt{a} \diamond \mathtt{y} \in G_3$
4: Define $\mathcal{X}_{(\mathtt{a},\mathtt{b})}$ to be the set of contexts common to the symbols $\mathtt{a}$ and $\mathtt{b}$, i.e., $\mathcal{X}_{(\mathtt{a},\mathtt{b})} = \mathcal{X}_{\mathtt{a}} \cap \mathcal{X}_{\mathtt{b}}$
5: Define $C_{\mathtt{xy}}(\mathtt{a}, \mathtt{b})$ to be the count of pair-wise combinations of symbols $\mathtt{a}$ and $\mathtt{b}$ in the given 3-gram context, $\mathtt{xy} \in \mathcal{X}_{(\mathtt{a},\mathtt{b})}$
   $$\begin{aligned} C_{\mathtt{xy}}(\mathtt{a}, \mathtt{b}) &= F_3(\mathtt{xay})F_3(\mathtt{xby}), \text{if } \mathtt{a} \neq \mathtt{b} \\ &= F_3(\mathtt{xay})(F_3(\mathtt{xay}) - 1)/2, \text{if } \mathtt{a} = \mathtt{b} \end{aligned}$$
6: Define $C(\mathtt{a}, \mathtt{b})$ to be the count of pair-wise combinations of the symbols $\mathtt{a}$ and $\mathtt{b}$ over all contexts $\mathcal{X}_{(\mathtt{a},\mathtt{b})}$
   $$C(\mathtt{a}, \mathtt{b}) = \sum_{\mathtt{xy} \in \mathcal{X}_{(\mathtt{a},\mathtt{b})}} C_{\mathtt{xy}}(\mathtt{a}, \mathtt{b})$$
7: Define $\mathcal{N}_C$ to be the norm of the count of pair-wise combinations
   $$\mathcal{N}_C = \sum_{\mathtt{a},\mathtt{b} \in \mathcal{A}} C(\mathtt{a}, \mathtt{b})$$
8: Define matrix $\mathcal{M}$ over $\mathcal{A} \times \mathcal{A}$ to be, $\mathcal{M}(\mathtt{a}, \mathtt{b}) = [C(\mathtt{a}, \mathtt{b})/\mathcal{N}_C]$
9: Define $p_{\mathtt{a}}$ to be the probability of occurrence of the symbol $\mathtt{a} \in \mathcal{A}$
   $$p_{\mathtt{a}} = \sum_{\mathtt{x} \in \mathcal{A}} \mathcal{M}(\mathtt{a}, \mathtt{x}); \quad \sum_{\mathtt{x} \in \mathcal{A}} p_{\mathtt{x}} = 1$$
10: Define matrix $\mathbb{E}$ to be the expected probability of pairs of symbols over $\mathcal{A} \times \mathcal{A}$
   $$\begin{aligned} \mathbb{E}(\mathtt{a}, \mathtt{b}) &= [p_{\mathtt{a}}^2], \text{ if } \mathtt{a} = \mathtt{b} \\ &= [p_{\mathtt{a}} p_{\mathtt{b}}], \text{ otherwise} \end{aligned}$$
11: Define the matrix of substitution scores $\mathbb{S}$ over $\mathcal{A} \times \mathcal{A}$ to be the log-odds ratio
   $$\begin{aligned} \mathbb{S}(\mathtt{a}, \mathtt{b}) &= \log_2 \left( \frac{\mathcal{M}(\mathtt{a}, \mathtt{b})}{\mathbb{E}(\mathtt{a}, \mathtt{b})} \right), \text{ if } \mathcal{M}(\mathtt{a}, \mathtt{b}) > 0 \\ &= -\log_2 \left( \frac{1}{\mathbb{E}(\mathtt{a}, \mathtt{b})} \right), \text{ if } \mathcal{M}(\mathtt{a}, \mathtt{b}) = 0 \end{aligned}$$
```

that $\mathtt{x} \diamond \mathtt{a} \diamond \mathtt{y} \in G_3$. In the above event log $\mathcal{L}$, the set of contexts of the symbol a is $\mathcal{X}_{\mathtt{a}} = \{\mathtt{aa, ab, ac, ad, bd, ca, db}\}$. Similarly, the set of contexts of the symbol b, $\mathcal{X}_{\mathtt{b}} = \{\mathtt{ac, bb, bc, ca, cb, db, dc, dd, ed}\}$ (Step 3, Algorithm 4.1). The common set of contexts for the symbols a and b is $\mathcal{X}_{(\mathtt{a,b})} = \{\mathtt{ac, ca, db}\}$ (Step 4, Algorithm 4.1).

Symbols that share a context can be favored to be substituted or aligned together. Each occurrence of a symbol in a particular context can potentially be aligned with every other occurrence of a symbol in the same context. We estimate the number of such possibilities for every pair of symbols in the given event log, which constitute the observed frequencies. Step 5 of Algorithm 4.1 computes the number of pair-wise combinations between two symbols having common contexts. To compute the pair-wise combinations of two symbols a and b within a particular context db, we need to consider the 3-grams with db as the context for symbols a and b, i.e., the 3-grams dab and dbb, and their frequencies. We have 3 occurrences of dab and 2 occurrences of dbb in the event log. Each occurrence of a can be aligned with each occurrence of b in the context db as shown in Figure 4.8(a). The number of such combinations for this case is $C_{\mathtt{db}}(\mathtt{a},\mathtt{b}) = 6$. Similarly, to calculate $C_{\mathtt{db}}(\mathtt{a},\mathtt{a})$, we need to consider the 3-gram dab. There are 3 occurrences of dab in the event log and each occurrence of a in the context of db can be aligned with every other occurrence of a other than itself as shown in Figure 4.8(b). Thus, the number of pair-wise combinations for this case is $C_{\mathtt{db}}(\mathtt{a},\mathtt{a}) = 3$.

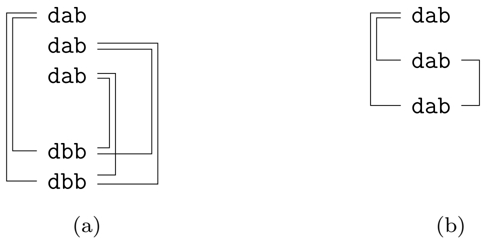


Figure 4.8: Pair-wise combinations of symbols with the same context. The count of pair-wise combinations $C_{\mathtt{db}}(\mathtt{a},\mathtt{b}) = 3.2 = 6$ and $C_{\mathtt{db}}(\mathtt{a},\mathtt{a}) = 3.(3-1)/2 = 3$.

In general, if the estimation of pair-wise combinations is for 'like' symbols, then the count of such combinations $C_{\mathtt{xy}}(\mathtt{a},\mathtt{a}) = \binom{n}{2} = \frac{n(n-1)}{2}$, where n is the frequency of the 3-gram xay. The count of pair-wise combinations for 'unlike' symbols $C_{\mathtt{xy}}(\mathtt{a},\mathtt{b}) = mn$ where m and n correspond to the frequency of the 3-grams xay and xby respectively.

Proceeding further, we can estimate the count of pair-wise combinations of two symbols over all contexts thereby completing Step 6 of the algorithm. Steps 7 and 8 of the algorithm normalize the counts thus calculated for every pair of symbols. Step 9 calculates the probability of occurrence of a particular symbol in the alphabet[5] while Step 10 calculates the normalized pair-wise combination frequencies that can

[5] $\sum_{\mathtt{x}\in\mathcal{A}} p_{\mathtt{x}} = \sum_{\mathtt{x}\in\mathcal{A}} \sum_{\mathtt{y}\in\mathcal{A}} \mathcal{M}(\mathtt{x},\mathtt{y}) = 1/\mathcal{N}_C \sum_{\mathtt{x},\mathtt{y}} C(\mathtt{x},\mathtt{y}) = \mathcal{N}_C/\mathcal{N}_C = 1$.

happen by chance (random). Step 11 computes the ratio of the actual frequency and the chance frequency with which the pair occurs. Such a ratio compares the probability of an event occurring under two alternative hypotheses and is called a likelihood or odds ratio. Scores that are the logarithm of odds ratios are called the log-odds score. If the actual frequency is more than the frequency by chance (which implies some statistical significance), then we get a positive score thus encouraging the substitution and viceversa. If two symbols never share a context, i.e., the observed frequency is zero, then we should not encourage the substitution of these symbols. Therefore, we assign a negative score $(-\log\left(\frac{1}{\mathbb{E}(\mathtt{a},\mathtt{b})}\right))$.

Table 4.1 depicts the substitution scores for some of the activity pairs in the digital copier example derived using Algorithm 4.1. We have used a log with 100 traces and $40,995$ events to generate these scores. The activities `Send FTP` and `Send SMTP`, modeled using a choice construct (see Figure 3.1), have the same context in the process and therefore can be allowed to be substituted. This is reflected in a relatively high substitution score of 8 for these two activities. The scores for two other activity pairs, viz., (`FM Screening`, `AM Screening`) and (`Coat Light Toner on Drum`, `Coat Toner on Drum`) modeled using a choice construct follow on similar lines. The substitution scores for like activities have a relatively high score, e.g., substituting `Coat Light Toner on Drum` with itself takes a score of 17. In contrast, the substitution of activities that do not occur in the same context have a negative score, e.g., the activities `Apply Heat` and `Apply Pressure` are in parallel branches and do not share the same contexts and hence is not encouraged to be substituted. Similarly, the activity pairs `Drum Spin Start` and `Fusing Complete` takes a relatively high negative score of -6 because they occur in two different subprocesses, viz., developing and fusing. It is important to note that although the activities `Apply Heat` and `Apply Pressure` do not share the same context, they belong to the same subprocess, viz., `Fusing`. The scores for substituting these activities is relatively better than that of `Drum Spin Start` and `Fusing Complete`, which pertain to completely different subprocesses. Thus, the substitution scores derived are in proportional to their degree of similarity. The negative scores just reflect the degree of dissimilarity. Since Equation 4.3 involves only the addition of scores, one can easily shift the substitution scores to make them all positive (just add $|\delta|$, i.e., the absolute value of δ, to all the scores where δ is the minimum of the scores over all activity pairs).

Table 4.1: Substitution scores for a few activity pairs of the digital copier example.

Activity 1	Activity 2	Substitution Score
Send FTP	Send SMTP	8
FM Screening	AM Screening	4
Coat Light Toner on Drum	Coat Light Toner on Drum	17
Coat Toner on Drum	Coat Light Toner on Drum	1
Apply Heat	Apply Pressure	-1
Drum Spin Start	Fusing Complete	-6

4.4.2 Insertion Scores

Just like substitution, insertion/deletion of activities cannot take place at random. It is natural to see insertion of activities pertaining to a functionality between activities related to the same or similar functionality than otherwise. Even within a functionality, the presence/absence of an activity largely depends on its neighbors. For example, it is highly unlikely to see an image processing activity between activities pertaining to printing (the image should have been processed before the printing can happen). Therefore, one should have different scores for insertion of activities based on the context.

We define two kinds of insertion operations: (i) insertion of an activity to the right of an activity and (ii) insertion of an activity to the left of an activity. For example, in the activity subsequence `abc`, activity `b` can be considered as an insertion to the right of activity `a` or to the left of activity `c`. We now define an approach to determine the scores of insertion. We define two sets of scores

a. Insertion Right Given Left, $\mathcal{I}_R : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$. $\mathcal{I}_R(\mathtt{a}, \mathtt{b})$ is the score of inserting activity `b` to the right of activity `a`.

b. Insertion Left Given Right, $\mathcal{I}_L : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$. $\mathcal{I}_L(\mathtt{a}, \mathtt{b})$ is the score of inserting activity `a` to the left of activity `b`.

Algorithm 4.2 generates the scores for the insertion of activities to the right of another activity, i.e., $\mathcal{I}_R$, and is straightforward following on similar lines to that of Algorithm 4.1. The insertion scores for $\mathcal{I}_L$ can be derived in a similar fashion.

Algorithm 4.2 Algorithm to derive insertion scores

```
1: Let $\mathcal{A}$ be the alphabet; $\mathtt{x}, \mathtt{y}, \mathtt{a}, \mathtt{b} \in \mathcal{A}$
2: Let $G_3$ denote the set of all 3-grams present in the event log $\mathcal{L}$, and let $F_3 : G_3 \rightarrow \mathbb{N}$
   denote their corresponding frequencies
3: Define $\mathcal{X}_\mathtt{a}$ to be the set of all contexts of the symbol a. A context of a symbol a
   is the subsequence xy ($\mathtt{x} \diamond \mathtt{y}$) such that $\mathtt{x} \diamond \mathtt{a} \diamond \mathtt{y} \in G_3$
4: For every pair of symbols $\mathtt{a}, \mathtt{x} \in \mathcal{A}$, let $C_R(\mathtt{x}, \mathtt{a})$ denote the frequency of occurrence
   of the symbol a to the right of x
       $$C_R(\mathtt{x}, \mathtt{a}) = \sum_{\mathtt{y} | \mathtt{xy} \in \mathcal{X}_\mathtt{a}} F_3(\mathtt{xay})$$
5: Define the normalization factor, $N(\mathtt{a}) = \sum_{\mathtt{x} \in \mathcal{A}} C_R(\mathtt{x}, \mathtt{a})$
6: For all $\mathtt{a} \in \mathcal{A}$, let $p_\mathtt{a}$ denote the probability of occurrence of a
7: The insertion scores are defined as the log-odds ratio
       $$\begin{aligned} \mathcal{I}_R(\mathtt{x}, \mathtt{a}) &= \log_2 \left( \frac{C_R(\mathtt{x}, \mathtt{a})/N(\mathtt{a})}{p_\mathtt{a} p_\mathtt{x}} \right), \text{ if } C_R(\mathtt{x}, \mathtt{a}) > 0 \\ &= -\log_2 \left( \frac{1}{p_\mathtt{a} p_\mathtt{x}} \right), \text{otherwise} \end{aligned}$$
```

Table 4.2 depicts the insertion scores (for insertion of Activity 2 to the right of Activity 1) for a few activity pairs of the digital copier example generated using an event log containing 100 traces and 40,995 events. Since `Send SMTP` is one of the two activities that can immediately follow `Transfer Image` in the process (cf. Figure 3.1), a relatively high score of 20 is assigned. The activities `X-Zoom` and `Y-Zoom` are modeled using a parallel construct in the Image Processing subprocess (cf. Figure 3.4). Hence either of the activities can follow each other and accordingly the score for inserting `Y-Zoom` after `X-zoom` takes a positive score. In contrast, the activities `FM Screening` and `AM Screening` are modeled using a choice construct and can never co-occur. Hence it is discouraged to insert `AM Screening` after `FM Screening` by assigning a high negative score.

Table 4.2: Insertion scores for a few activity pairs of the digital copier example.

Activity 1	Activity 2	Insertion Right Given Left Score
Transfer Image	Send SMTP	20
X-Zoom	Y-Zoom	16
Coat Toner on Drum	Drum Spin Stop	3
Apply Heat	Heated Roller Spin Stop	6
AM Screening	Screening Complete	11
FM Screening	AM Screening	-12

The algorithms defined above derive scores for substitution/indel operations such that similar traces attain a high score. One can convert a similarity measure into a distance measure. The scores for substitution and indel derived using the approaches presented in Sections 4.4.1 and 4.4.2 can be negative and, thus, there is a possibility for the similarity measure computed using Equation (4.3) to be negative. However, the distance measures should be positive. We can easily ensure this by *shifting* the derived substitution and indel scores by adding $|s_{\min}|$ to the substitution scores of all activity pairs (where $s_{\min}$ is the minimum of the substitution scores) and $|i_{\min}|$ to the indel scores of all activity pairs (where $i_{\min}$ is the minimum of the indel scores). One can compute the similarity between traces using these updated scores and take the reciprocal of that as a measure of distance, i.e., for two traces $\mathbf{s}$ and $\mathbf{t}$, the generic edit distance, *ged*, between them can be defined as

$$ged(\mathbf{s},\mathbf{t}) = \frac{|\mathbf{s}| + |\mathbf{t}|}{sim(\mathbf{s},\mathbf{t})}$$

where the numerator denotes the normalization factor [263]. However, other normalizations [149, 273] can also be adopted. Since edit distance is influenced by the lengths of the traces, we use the above normalization scheme to compensate for scores contributed by the variation in lengths.

Having discussed ways of estimating the (dis-)similarity between traces, let us now look at algorithms for clustering.

4.5 Algorithms for Clustering

Given an event log containing a bag of traces and a selected distance/similarity function, the objective of clustering is to partition the event log into bags of homogenous traces (for a chosen number of clusters/partitions, $k \in \mathbb{N}$). To achieve this, many clustering algorithms exist [104, 113, 114]. k-means clustering, hierarchical clustering, and self-organizing maps [129] are some of the commonly used techniques. Here, we focus on the hierarchical clustering algorithm.

The hierarchical clustering algorithm initially places all data elements in single clusters and then merges these clusters into larger clusters in a bottom-up fashion. Different criteria can be used during this merging process, e.g., single linkage, complete linkage, minimum variance, etc. [113]. The agglomerative hierarchical clustering (AHC) algorithm is informally described in Algorithm 4.3.

Algorithm 4.3 Agglomerative Hierarchical Clustering

```
1: Determine all inter-object dissimilarities, i.e., determine the distance between
   every pair of traces
2: Form a cluster from two closest traces or clusters
3: Redefine dissimilarities between new cluster and other traces or clusters (all other
   inter-object dissimilarities remain unchanged)
4: Return to Step 2 until all traces are in one cluster
```

Figure 4.9 illustrates an example of applying this on five traces using the single linkage and complete linkage join criteria. At the top of the figure is the distance matrix between the traces (one can use either the vector-based approaches or the syntactic approaches to obtain this matrix). Since there are five traces, four iterations of the algorithm are needed. In this example, the traces $[\mathbf{t}_2]$ and $[\mathbf{t}_3]$ are the most similar (have the least distance) and hence clustered first. The distance between the newly formed cluster $G_1 = [\mathbf{t}_2, \mathbf{t}_3]$ and the rest of the traces are updated. The updations are guided by the join criteria. For example, let $[p]$ and $[q]$ be the two objects that are grouped together in this iteration, denoted by [p, q]; the distance between an object $[r]$ and the newly formed cluster is updated as $dist([r],[p,q]) = \min\{dist([r],[p]), dist([r],[q])\}$ in the single linkage criteria and as $dist([r],[p,q]) = \max\{dist([r],[p]), dist([r],[q])\}$ in the complete linkage criteria. Thus, the distance between the trace $[\mathbf{t}_1]$ and the cluster $G_1 = [\mathbf{t}_2, \mathbf{t}_3]$ is updated to 1.4 ($\min\{1.4, 2.1\}$) and 2.1 ($\max\{1.4, 2.1\}$) respectively under the single linkage and complete linkage criteria. The updated distance matrices according to both these criteria are as illustrated in Figure 4.9.

In the next iteration, the traces $[\mathbf{t}_4]$ and $[\mathbf{t}_5]$ are the most similar and hence

$$\begin{array}{c|ccccc} & [\mathbf{t}_1] & [\mathbf{t}_2] & [\mathbf{t}_3] & [\mathbf{t}_4] & [\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & 1.4 & 2.1 & 2.8 & 3.2 \\ [\mathbf{t}_2] & & 0 & \boxed{0.2} & 1.6 & 1.8 \\ [\mathbf{t}_3] & & & 0 & 1.7 & 2.0 \\ [\mathbf{t}_4] & & & & 0 & 0.8 \\ [\mathbf{t}_5] & & & & & 0 \end{array}$$

single linkage

$$\begin{array}{c|cccc} & [\mathbf{t}_1] & [\mathbf{t}_2,\mathbf{t}_3] & [\mathbf{t}_4] & [\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & 1.4 & 2.8 & 3.2 \\ [\mathbf{t}_2,\mathbf{t}_3] & & 0 & 1.6 & 1.8 \\ [\mathbf{t}_4] & & & 0 & \boxed{0.8} \\ [\mathbf{t}_5] & & & & 0 \end{array}$$

$$\begin{array}{c|ccc} & [\mathbf{t}_1] & [\mathbf{t}_2,\mathbf{t}_3] & [\mathbf{t}_4,\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & \boxed{1.4} & 2.8 \\ [\mathbf{t}_2,\mathbf{t}_3] & & 0 & 1.6 \\ [\mathbf{t}_4,\mathbf{t}_5] & & & 0 \end{array}$$

$$\begin{array}{c|cc} & [\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}_3] & [\mathbf{t}_4,\mathbf{t}_5] \\ \hline [\mathbf{t}_1,\mathbf{t}_2,\mathbf{t}_3] & 0 & \boxed{1.6} \\ [\mathbf{t}_4,\mathbf{t}_5] & & 0 \end{array}$$

complete linkage

$$\begin{array}{c|cccc} & [\mathbf{t}_1] & [\mathbf{t}_2,\mathbf{t}_3] & [\mathbf{t}_4] & [\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & 2.1 & 2.8 & 3.2 \\ [\mathbf{t}_2,\mathbf{t}_3] & & 0 & 1.7 & 2.0 \\ [\mathbf{t}_4] & & & 0 & \boxed{0.8} \\ [\mathbf{t}_5] & & & & 0 \end{array}$$

$$\begin{array}{c|ccc} & [\mathbf{t}_1] & [\mathbf{t}_2,\mathbf{t}_3] & [\mathbf{t}_4,\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & 2.1 & 3.2 \\ [\mathbf{t}_2,\mathbf{t}_3] & & 0 & \boxed{2.0} \\ [\mathbf{t}_4,\mathbf{t}_5] & & & 0 \end{array}$$

$$\begin{array}{c|cc} & [\mathbf{t}_1] & [\mathbf{t}_2,\mathbf{t}_3,\mathbf{t}_4,\mathbf{t}_5] \\ \hline [\mathbf{t}_1] & 0 & \boxed{3.2} \\ [\mathbf{t}_2,\mathbf{t}_3,\mathbf{t}_4,\mathbf{t}_5] & & 0 \end{array}$$

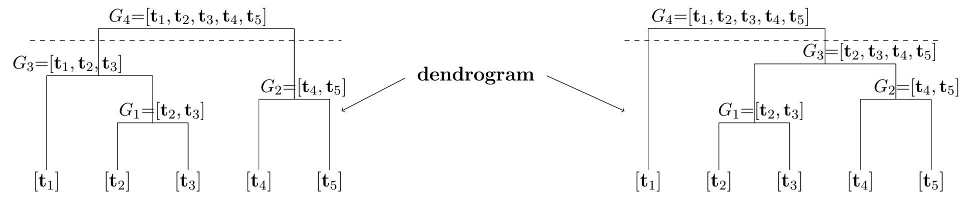


Figure 4.9: Example of distance matrix updations using the single linkage and complete linkage methods. The cell in the distance matrix corresponding to the pair of objects that are the most similar in an iteration is encircled.

grouped together, in both the single linkage and complete linkage criteria. The distance between this newly formed cluster, $G_2 = [\mathbf{t}_4, \mathbf{t}_5]$, and the other objects, viz., cluster $G_1 = [\mathbf{t}_2, \mathbf{t}_3]$, and the trace $[\mathbf{t}_1]$ is to be updated. The updated matrices are as illustrated in Figure 4.9. In the third iteration, under the single linkage criteria, the objects $G_1 = [\mathbf{t}_2, \mathbf{t}_3]$ and the trace $[\mathbf{t}_1]$ are the most similar, and hence are grouped together. However, under the complete linkage criteria, the objects $G_1 = [\mathbf{t}_2, \mathbf{t}_3]$ and $G_2 = [\mathbf{t}_4, \mathbf{t}_5]$ have the least distance. Therefore, we get two different groupings under these two criteria, viz., $G_3 = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ under the single linkage criteria and $G_3 = [\mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4, \mathbf{t}_5]$ under the complete linkage criteria. Proceeding futher, the distance matrices are updated. In the fourth (and final) iteration, the two clusters G_3 and G_2 are grouped together under the single linkage criteria while the cluster G_3 is grouped with the trace $[\mathbf{t}_1]$ in the complete linkage criteria. The hierarchical grouping of objects can be visualized using a so-called *dendrogram* as shown in Figure 4.9. Concrete clusters can be obtained by cutting through the dendrogram horizontally at any point. Figure 4.9 shows one such horizontal line that results in two clusters. The two clusters formed under the single linkage criteria are $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ and $[\mathbf{t}_4, \mathbf{t}_5]$. In contrast, complete linkage produces the two clusters as $[\mathbf{t}_1]$ and $[\mathbf{t}_2, \mathbf{t}_3, \mathbf{t}_4, \mathbf{t}_5]$. The number of clusters can be varied by moving the horizontal line.

We adopt the AHC algorithm for our studies because of its (a) embedded flexibility on the level of abstraction and (b) ease in handling any form of similarity or distance. k-means clustering, for example, generates only a single partition and cannot be applied for syntactic methods based on edit distance.

Figure 4.10 depicts the framework for trace clustering. The framework identifies the following steps:

- *Feature Extraction and Selection:* This is applicable for the vector-based approaches to trace clustering and corresponds to the transformation of traces into vector space. We can choose one or more (union) of the feature sets presented in Section 4.2. In addition, we can choose to filter features that are insignificant (e.g., features that are less frequent) during this transformation process.
- *Compute Scoring Matrices:* As discussed earlier, syntactic approaches based on edit distance require a cost/scoring function. This step corresponds to the derivation of scores for substitution and insertion/deletion using the techniques presented in Section 4.4. For the Levenshtein distance, a unit scoring function is used.
- *Compute Distance or Similarity:* This step corresponds to the estimation of distance or similarity between every pair of traces. Distance metrics such as the Euclidean distance and Mahalanobis distance [144] and similarity metrics such as the F-Score [62] can be used for the vector-based approaches. For the syntactic approaches, one can use measures such as the Levenshtein edit distance and the generic edit distance.
- *Group Objects:* This step corresponds to the grouping of traces into clusters. We adopt the Agglomerative Hierarchical Clustering algorithm as discussed earlier for grouping traces.
- *Interactive Visualization:* The AHC algorithm generates a hierarchy of clusters that can be visualized as a dendrogram. This step corresponds to providing interactive means of visualizing and exploring the dendrogram. The dendrogram can be annotated with additional information characterizing the traces grouped under a node.

4.6 Computational Complexity

We now discuss the computational complexity of the AHC algorithm. The AHC algorithm involves the construction of $n-1$ clusters, and so $n-1$ iterations (Steps 2, 3, and 4 of Algorithm 4.3). Step 1 requires $\mathcal{O}(n^2)$ calculations, i.e., $n(n-1)/2$

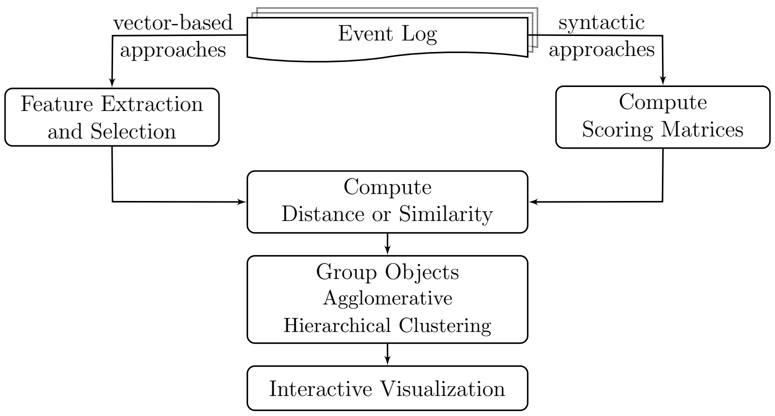

vector-based approaches
Event Log
syntactic approaches
Feature Extraction and Selection
Compute Scoring Matrices
Compute Distance or Similarity
Group Objects Agglomerative Hierarchical Clustering
Interactive Visualization

Figure 4.10: Framework for trace clustering.

inter-trace dissimilarities, where n is the number of traces. As mentioned earlier, the distance between the traces can be computed either by defining some features and transforming each trace into a vector space or by using syntactic edit distance approaches. Edit distance approaches are time consuming in that it takes quadratic time (with respect to the length of the traces) to compute the distance between a pair of traces, i.e., $\mathcal{O}(pq)$ where p and q are the lengths of the two traces. In the vector-space approach, the number of variables (features) obviously affects the calculation time required, but they are usually considered constant for any set of data, e.g., computing the Euclidean distance between two traces varies linearly to the number of features. A naive attempt at Step 2 will make the algorithm $\mathcal{O}(n^3)$ complex as the minimum of an $n \times n$ matrix must be found in each of the $n - 1$ iterations. However, this can be reduced to $\mathcal{O}(n^2)$ by maintaining a pointer to the minimum value in each row of the matrix. Step 3 can be carried out in $\mathcal{O}(n)$ time for each iteration using the Lance–Williams combinatorial formula [133, 158]. Thus, the overall complexity of the AHC algorithm is quadratic, i.e., $\mathcal{O}(n^2)$, with respect to the number of traces.

4.7 Evaluating the Goodness of Clusters

The interpretation of the formed clusters is subjective in nature and the goodness of the resulting clusters is largely influenced by choice of the feature space, distance/similarity measures, and the clustering algorithm adopted. Statistical metrics such as the Dunn's index [60], silhouette width [188], etc., have been proposed in the literature to evaluate the goodness of the clusters. The underlying motive for these metrics is to prefer clusters that are compact and well separated; such metrics are categorized as *internal metrics*. Compact clusters have a lot of significance in pattern classification where the objective is to enable the discovery of decision boundaries between classes of data. Clusters with good scores for such metrics do not necessarily imply effective clusters in an application. An alternative set of metrics, called as *external metrics*, employs external class information called *gold standard classes*.

The gold standard on the class association of each data element is to be given ideally by a domain expert. Once such external class information is available, one can use metrics such as the Rand Index [179] or supervised classification metrics such as the F-measure [104] to evaluate the goodness of clusters. The reader is referred to [100, 101] for a review of cluster validation metrics.

The objective for clustering event logs in process mining is to ease the discovery of process models and analysis of process execution behavior by grouping together traces that conform to similar execution patterns/behavior. To evaluate the significance of the clusters formed, one can compare the process models that are discovered from the traces within each cluster. We propose two hypotheses to evaluate the goodness of clusters from a process mining point of view. Good clusters tend to have traces such that:

1. the process models mined show a high degree of fitness, and
2. the process models mined have low complexity

The rationale behind these evaluation criteria is that if the clusters formed are meaningful (all traces belonging to related cases are in the same cluster and traces that are unrelated are not), then the process models resulting from the traces in each cluster should be less complex (more comprehensible and less spaghetti-like) and be able to describe the traces in the cluster better (which is captured by the fitness measure). The actual fitness measure adopted for the process model can vary based on the discovery algorithm, e.g., it could be the continuous semantics [49, 53] or the improved continuous semantics [49] measure for the Heuristic Miner [264]. We propose two derived fitness measures, viz., *average fitness* and *weighted average fitness* defined as follows:

- Average Fitness, f_{avg}, is defined as the average of the fitness of the process models mined from each of the clustered traces, i.e., $f_{\text{avg}} = \sum_{i=1}^{k} f_i / k$, where k is the number of clusters and f_i is the fitness of the process model mined from traces belonging to cluster i.
- Weighted Average Fitness, wf_{avg}, takes into account the number of traces in each of the clusters and is defined as $wf_{\text{avg}} = \sum_{i=1}^{k} n_i f_i / \sum_{i=1}^{k} n_i$, where k is the number of clusters, f_i is the fitness of the process model mined from traces belonging to cluster i, and n_i is the number of traces in cluster i.

Weighted average fitness balances the fitness over imbalanced clusters[6]. For example, consider the scenario where 100 traces are partitioned into four clusters with $n_1 = 5$, $n_2 = 6$, $n_3 = 9$, and $n_4 = 80$ instances. Assume that the process models mined from the first three clusters have a fitness value of 1.0 while the model from the fourth cluster has a fitness value of 0.8. The average fitness value would be $(1.0+1.0+1.0+0.8)/4 = 0.95$. However, the distribution of traces is skewed in the clusters and the average fitness value does not reflect the reality. The weighted average fitness value for this partitioning would be $(5\times1.0+6\times1.0+9\times1.0+80\times0.8)/100 = 0.84$, a better summarization of the goodness of clusters.

[6]scenarios where the partitioning of traces is skewed, i.e., the number of traces in some clusters is much less compared to the others.

Mendling et al. [152–154, 183] have discussed a comprehensive list of metrics that influence the understandability of process models. We propose three metrics, viz., *average number of nodes, average number of arcs*, and *average number of arcs per node* to assess the complexity of a process model based on their structural properties. The intuition is that clustering should enable the partitioning of traces based on functionality and that the resulting clusters should have event classes pertaining only to those events that constitute the functionality. Good clusters tend to form clusters such that the number of event classes is minimal per cluster. The average number of arcs and average number of arcs per node are measures of spaghettiness of the process model.

4.8 Experiments and Discussion

In this section, we report the experimental results of the various approaches presented in this chapter. We simulated multiple events logs for the digital photo copier using CPN tools [180]. The copier has two primary functions, viz., copy/scan and printing of documents. The two functionalities have enough discriminatory activities to clearly differentiate them. Any clustering technique on the event log would easily segregate the two classes when asked to partition a log into two clusters. To study the effectiveness of the techniques proposed in this chapter, we consider only those cases pertaining to copy/scan job requests. We add noise in some of the copy/scan cases by randomly reordering certain activities. We use one such event log (after adding noise) that contains 116 cases and $10,560$ events referring to 35 activities. 97 of these cases are valid (V) while 19 cases are outliers/noisy (O). Figure 4.11 depicts the Petri net mined using the α-algorithm [229] implemented in ProM 5.2. The copy/scan subprocess is a well-structured process but the presence of outliers disturbs the discovery algorithm leading to an incomprehensible spaghetti-like process. We evaluate the goodness of the context-aware approaches and compare them with contemporary approaches to trace clustering using this event log. *Ideally, when we split the event log into two clusters, we expect the valid and outlier cases to be segregated in the two clusters.*

Figure 4.12 depicts the transposed matching matrix (confusion matrix)[7] for the various feature sets and the syntactic methods. The event log was split into two clusters using the Agglomerative Hierarchical Clustering (AHC) algorithm with the minimum variance [259] as the join criteria. For the vector based feature sets, Euclidean distance was adopted to define the distance between the traces. The formed clusters are assigned labels such that the sum of the values in the leading diagonal is maximum. Figure 4.12 also depicts the results of clustering using the sequence clustering algorithm [70, 249] (see column SeqClus). We can see that the best performance is achieved by the process-centric feature sets MRA and NSMRA, where

[7]The result of each feature set/technique is denoted by a 2×2 rectangular matrix. The value in a cell (i, j) under a particular feature set/technique denotes the number of traces of the class labeled in column j that are assigned to a cluster labeled with the class in row i.

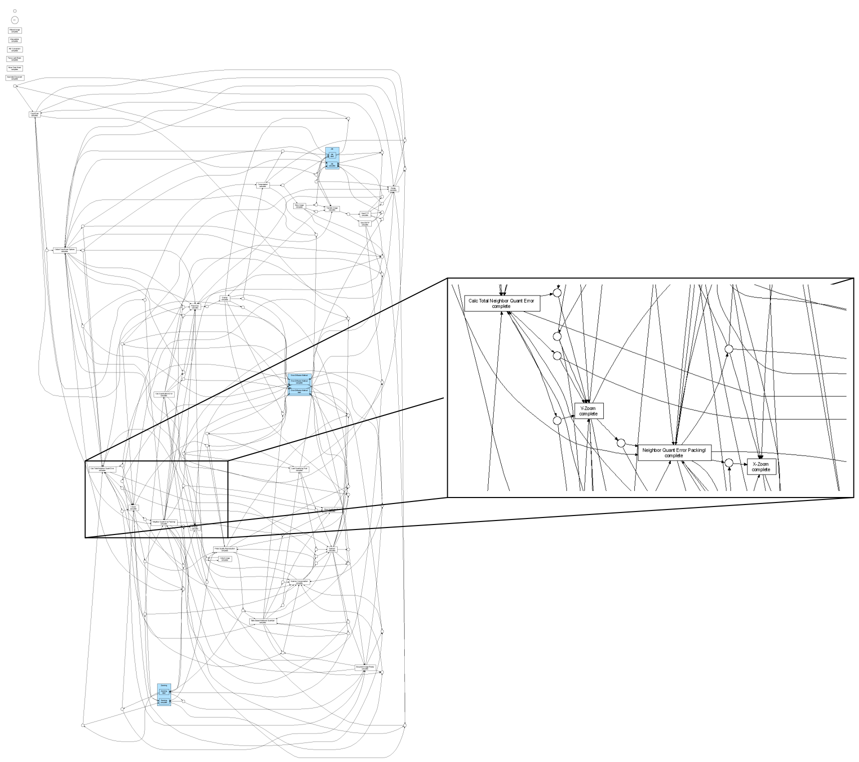

Calc Total Neighbor Quant Error complete
Y-Zoom complete
Neighbor Quant Error Packing! complete
X-Zoom complete

Figure 4.11: Petri net mined using the α-miner plug-in in ProM on the event log containing a mix of valid and outlier traces.

a perfect segregation of valid and outlier traces is obtained. The worst performance is achieved by the super maximal repeat alphabet feature set, SMRA. This can be attributed to the fact that there were only 125 SMRA features, the majority of which are features involving just two activities that do not possess any discriminatory information. In contrast, there were 278 NSMRA features. Recall that super maximal repeats are those maximal repeats that are not subsumed in any other maximal repeat. Discriminatory maximal repeats that are subsumed in less frequent longer maximal repeats are all ignored thereby leading to the poor performance of SMRA. The k-gram, MR, NSMR and SMR feature sets are able to form clusters such that at least one cluster contains only the valid traces with the other containing all the outlier traces in addition to some valid traces. For the rest of the feature sets, the outlier traces are distributed across both the traces.

For the sequence clustering approach [70, 249] we use the implementation available as the ‘Sequence Clustering’ plug-in in ProM 5.2. The plug-in behaves non-deterministically in that the formed clusters are sensitive to the random initialization of the clusters. This leads to different partitionings in different iterations. The values provided in Figure 4.12 are the best out of five independent runs. Although sequence

	BOA		kGram		LED		GED		MR		NSMR		SMR		MRA ✓		NSMRA ✓		SMRA		SeqClus	
	O	V	O	V	O	V	O	V	O	V	O	V	O	V	O	V	O	V	O	V	O	V
C1 (O)	4	27	19	46	7	42	5	26	19	8	19	8	19	13	**19**	**0**	**19**	**0**	0	1	15	0
C2 (V)	15	70	0	51	12	55	14	71	0	89	0	89	0	84	**0**	**97**	**0**	**97**	19	96	4	97

Figure 4.12: Transposed matching matrix obtained using AHC on the various feature sets and syntactic methods, and using the sequence clustering algorithm. The feature sets marked with a tick mark are able to assist in obtaining a perfect segregation of valid and outlier traces.

clustering is unable to produce a perfect partitioning, it forms reasonable clusters that are able to segregate most of the outlier traces. We could not assess the performance of outlier detection techniques [72, 80] due to inaccessibility of their software implementation (the authors mentioned that the software is currently proprietary).

Figure 4.13 depicts the Petri net mined using the α-algorithm on the traces captured in the valid cluster using the MRA feature set (recall that MRA feature set leads to a perfect segregation of valid and outlier traces). Thus by segregating homogenous cases, we are able to obtain a comprehensible model that can be analyzed further. Unlike the α-algorithm, the Heuristic miner [264] is relatively more robust to noise and is able to discover structured processes on this event log even in the presence of outliers, albeit with a low fitness. Figure 4.14 depicts the heuristic net mined using the Heuristic miner on the original event log (with outliers). Figure 4.14 is well-structured when compared to Figure 4.11.

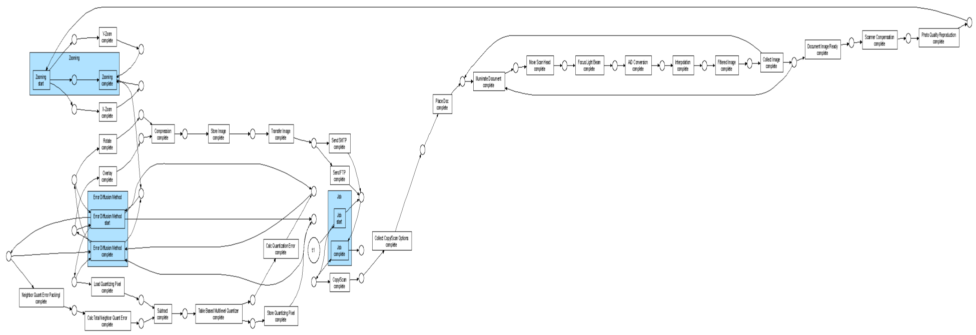

Y-Zoom complete
Zooming
Zooming start
Zooming complete
X-Zoom complete
Rotate complete
Overlay complete
Compression complete
Store Image complete
Transfer Image complete
Send SMTP complete
Send FTP complete
Error Diffusion Method
Error Diffusion Method start
Error Diffusion Method complete
Job
Job start
Job complete
t1
Calc Quantization Error complete
Collect Copy/Scan Options complete
Copy/Scan complete
Neighbor Quant Error Packing! complete
Load Quantizing Pixel complete
Calc Total Neighbor Quant Error complete
Subtract complete
Table Based Multilevel Quantizer complete
Store Quantizing Pixel complete
Place Doc complete
Illuminate Document complete
Move Scan Head complete
Focus Light Beam complete
AD Conversion complete
Interpolation complete
Filtered Image complete
Collect Image complete
Document Image Ready complete
Scanner Compensation complete
Photo Quality Reproduction complete

Figure 4.13: Petri net mined on the valid cluster generated by the MRA feature set using the α-algorithm.

Figure 4.15(a) depicts the *continuous semantics* [49, 53] measure computed on the process model mined using the Heuristic miner [264] on the traces belonging to the valid cluster[8]. We can see that techniques that consider contexts (k-grams, process-centric feature sets, edit distance, and sequence clustering) largely outperform the BOA feature set that does not capture any context information. Figure 4.15(a) also depicts the weighted continuous semantics measure computed on the process models mined using both the clusters. It can be seen that the context-aware approaches outperform the BOA feature set. Moreover, the process-centric feature sets perform better than the k-gram, edit distance, and sequence clustering. Within the syntactic approaches to trace clustering, the generic edit distance performs better than the Levenshtein distance. This is to be expected because the generic edit distance uses a cost function that is sensitive to the activities unlike the Levenshtein distance, which uses a unit cost function (cf. Section 4.3).

[8]Since the super maximal repeat alphabet feature set almost retained the event log as is, we did not consider that feature.

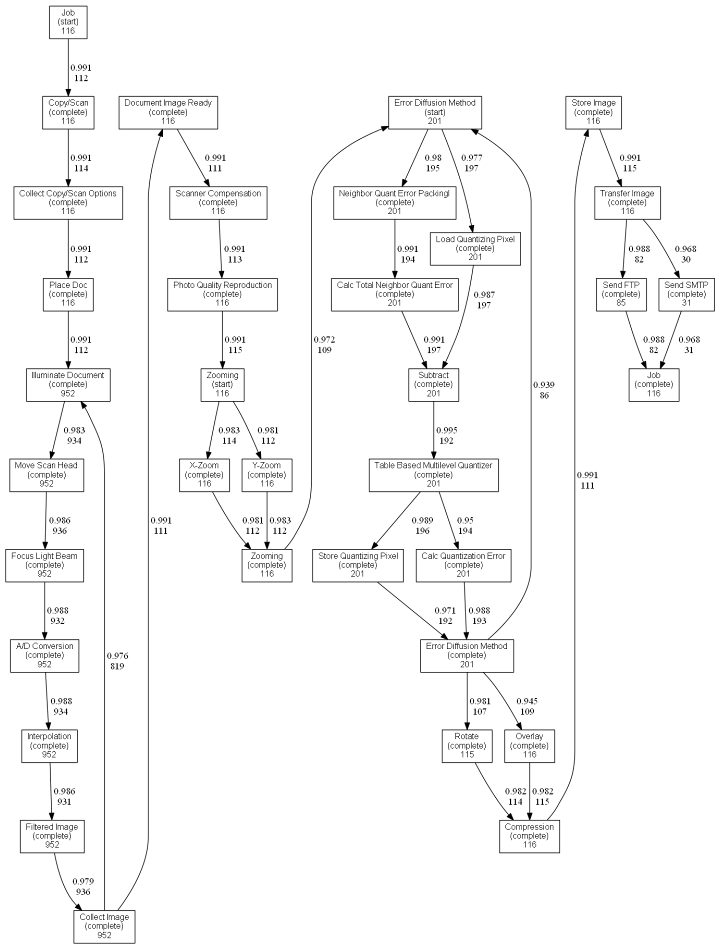
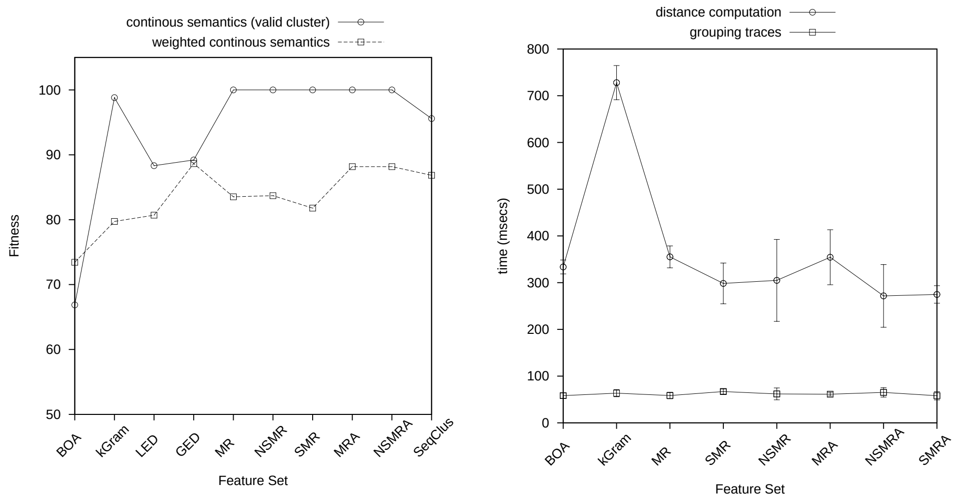

Job (start) 116
0.991 112
Copy/Scan (complete) 116
0.991 114
Collect Copy/Scan Options (complete) 116
0.991 112
Place Doc (complete) 116
0.991 112
Illuminate Document (complete) 952
0.983 934
Move Scan Head (complete) 952
0.986 936
Focus Light Beam (complete) 952
0.988 932
A/D Conversion (complete) 952
0.988 934
Interpolation (complete) 952
0.986 931
Filtered Image (complete) 952
0.979 936
Collect Image (complete) 952
0.976 819
0.991 111
Document Image Ready (complete) 116
0.991 111
Scanner Compensation (complete) 116
0.991 113
Photo Quality Reproduction (complete) 116
0.991 115
Zooming (start) 116
0.983 114
0.981 112
X-Zoom (complete) 116
Y-Zoom (complete) 116
0.981 112
0.983 112
Zooming (complete) 116
0.972 109
Error Diffusion Method (start) 201
0.98 195
0.977 197
Neighbor Quant Error Packingl (complete) 201
0.991 194
Load Quantizing Pixel (complete) 201
Calc Total Neighbor Quant Error (complete) 201
0.987 197
0.991 197
Subtract (complete) 201
0.995 192
Table Based Multilevel Quantizer (complete) 201
0.989 196
0.95 194
Store Quantizing Pixel (complete) 201
Calc Quantization Error (complete) 201
0.971 192
0.988 193
Error Diffusion Method (complete) 201
0.939 86
0.981 107
0.945 109
Rotate (complete) 115
Overlay (complete) 116
0.982 114
0.982 115
Compression (complete) 116
0.991 111
Store Image (complete) 116
0.991 115
Transfer Image (complete) 116
0.988 82
0.968 30
Send FTP (complete) 85
Send SMTP (complete) 31
0.988 82
0.968 31
Job (complete) 116

Figure 4.14: Heuristic net mined on the event log with outliers.

Computational Complexity and Scalability Analysis

The total time required to cluster an event log using AHC algorithm can be divided into the time to (a) extract the features (for vector-based approaches), (b) compute the inter-trace distance/similarity (Step 1 of Algorithm 4.3), and (c) form the hierarchy, i.e., grouping of objects (Steps 2, 3, and 4 of Algorithm 4.3). The process-centric feature sets can all be efficiently discovered in linear time using suffix trees as discussed in Chapter 3. Figure 4.15(b) depicts the average time[9] (along with the 95% confidence intervals over five independent runs) required to compute the distance/similarity between all pairs of traces and the grouping of objects using AHC algorithm. It can be seen that using the process-centric feature sets is computationally efficient too. As discussed in Section 4.2, the *k*-gram feature set may

[9]All the computational times reported in this chapter are measured on an i3 Core CPU M350 @ 2.27 GHz with 4GB RAM running a 64-bit Windows 7 OS.



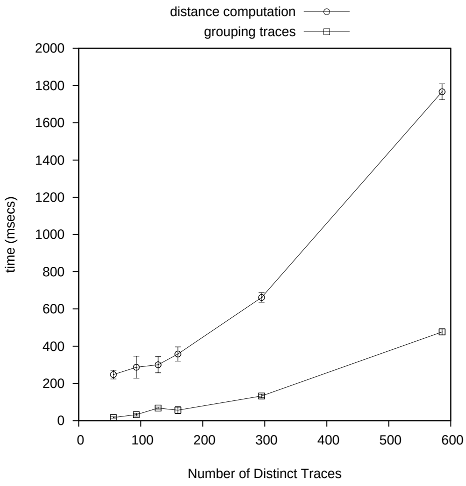
(a) fitness (b) computational time

Figure 4.15: Fitness as a measure of cluster efficiency and computational time required to form the clusters using AHC algorithm. The process models are discovered using the Heuristic miner [264] and continuous semantics metric [49, 53] is adopted as the fitness measure. 95% confidence intervals are also depicted for the computation time.

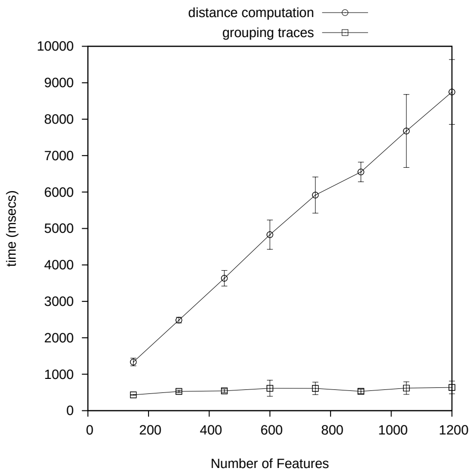
lead to an excessive number of features as the number of activities and the value of k increase. The time to compute the distance between every pair of traces in the vector-based approach is proportional to the number of features. The syntactic approaches to computing the distance/similarity between the traces is computationally expensive as it takes quadratic time to compute the distance between every pair of traces (with respect to the length of the traces). The average time (over 5 independent runs) required to compute the Levenshtein distance between all pairs of traces was 1595 ± 42 msecs while the generic edit distance took 26542 ± 5745 msecs. The generic edit distance takes relatively more time due to the fact that the contexts need to be considered at each edit operation. The grouping of objects only depends on the number of traces; therefore, the time to group objects is the same for all the feature sets. Furthermore, the grouping of objects takes just a fraction of the time required to compute the distance. The sequence clustering algorithm took 3352 ± 1205 msecs.

4.8.1 Influence of the Number of Traces and Feature Set Size

We now study the influence of the number of traces and the number of features on the computational time. We have simulated multiple event logs with varying number of cases for the digital copier example and considered the copy/scan jobs. In order to isolate the influence of the number of traces, we keep the feature set and the number of features constant. For this study, we chose the maximal repeat alphabet (MRA) as the feature set and considered only the top 200 (most frequent) features. Figure 4.16(a) depicts the influence of the number of traces on computing the distance and the grouping of traces. The average time taken over 5 independent runs along with the 95% confidence intervals is depicted in Figure 4.16(a). As expected, the distance computation varies quadratically with respect to the number of traces. The grouping of traces varies sub-quadratically. In order to study the influence of the feature set size, we considered a log containing 586 distinct traces and 58, 665 events distributed over 35 event classes. We chose the MRA feature set and varied the number of features by considering only the top 'q' frequent features. Figure 4.16(b) depicts the average time (along with the 95% confidence intervals over 5 independent runs) required to compute the Euclidean distance between all pairs of traces for varying number of features. As expected, computing the Euclidean distance varies linearly with respect to the number of features. However, one needs to exercise caution when removing features as this may impact the goodness of the resulting clusters.



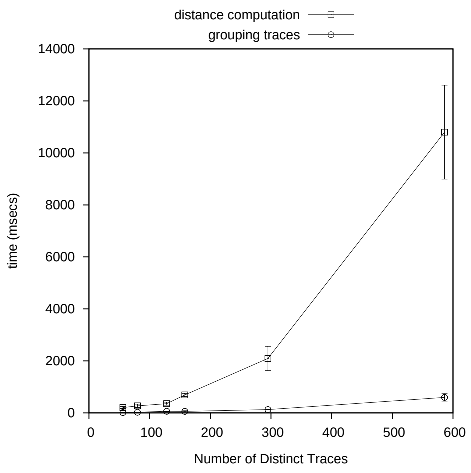
(a) Influence of the number of traces



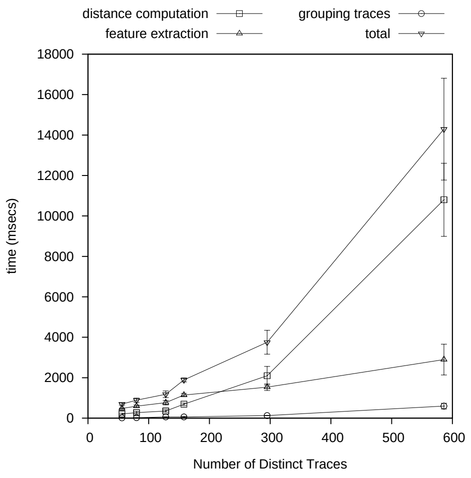
(b) Influence of the feature set size

Figure 4.16: Influence of the number of traces and the feature set size on distance computation.

In a realistic scenario as the size of the log changes, both the number of traces and the number of features may change. Figure 4.17(a) depicts the average time (along with the 95% confidence intervals over five independent runs) required to compute the distance/similarity between all pairs of traces and the grouping of objects using AHC algorithm for different logs in the realistic scenario using the MRA feature set. We can see that computing the distance varies quadratically with the number of traces while the grouping of traces varies sub-quadratically. Figure 4.17(b) depicts the feature extraction time for the different logs and the total time, which is basically the sum of the times taken for feature extraction, distance computation, and the grouping of traces, for clustering event logs. It can be seen that the feature extraction, which is the discovery of maximal repeats, varies linearly with the size of the log (number of events) as discussed in Chapter 3. The total time to cluster an



(a) Time to compute the distance and group objects



(b) Overall time to cluster

Figure 4.17: Computational time required to cluster different logs of varying size using the MRA feature set.

event log is dominated by the time to compute the similarity/distance between the traces.

Figure 4.18 depicts the time taken to cluster various event logs of different sizes for the AHC and sequence clustering [69, 70, 249] approaches. We considered the MRA feature set and the Euclidean distance with minimum variance as the join criteria for the AHC algorithm. The sequence clustering algorithm becomes prohibitively expensive as the number of traces increases. This, combined with the need for multiple runs of the algorithm to decide on good clusters (due to its non-deterministic behavior) makes it less attractive for large event logs.

4.8.2 Influence of the Number of Traces and Trace Length

The computation time of edit distance is sensitive to the length of the traces and varies quadratically with respect to the length of the traces. For an event log with n traces and an average trace length of l, the time complexity for computing the edit distance between all pairs of traces is $\mathcal{O}(l^2n^2)$. Figure 4.19(a) depicts the average time taken (over 5 independent runs) along with the 95% confidence intervals to compute the Levenshtein edit distance for varying number of traces. The average trace length is also depicted. The *polynomial variation of the computational time* can be noticed. Figure 4.19(b) depicts the average time along with the 95% confidence intervals taken (over 5 independent runs) to compute the generic edit distance for varying number of traces. The trend is as expected. However, it is to be noted that using the generic edit distance is much more expensive than that of the Levenshtein distance. This can be attributed to the fact that the contexts for each edit operation

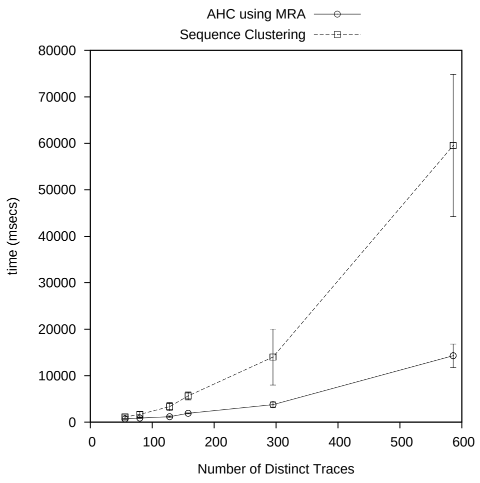
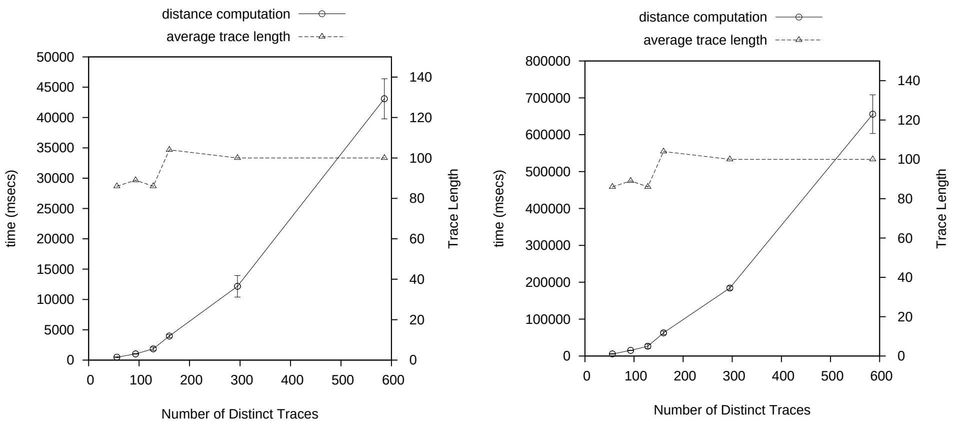


Figure 4.18: Comparison of the computational time required to cluster different logs of varying size using the AHC algorithm considering MRA as the feature set, and the sequence clustering algorithm.

need to be considered during the computation. Syntactic approaches take orders of magnitude longer than vector-based approaches and hence should only be preferred when the event logs are small and the traces are short.



(a) Levenshtein edit distance

(b) Generic edit distance

Figure 4.19: Computational time required to compute distance between the traces using syntactic approaches for different logs of varying size.

4.9 Extensions

In this chapter, we looked at trace clustering only from a control-flow point of view. We can extend the techniques proposed in this chapter with additional perspectives just like in [88, 209]. For the vector-based approaches, additional feature sets catering to the data, organizational, and time perspectives can be seamlessly combined with the process-centric feature sets proposed in this chapter. Furthermore, the process-centric feature sets can be directly applied on traces formed by considering the resource attribute as a classifier, i.e., traces correspond to the sequence of resources rather than activities. One can either use traditional clustering techniques such as the AHC and k-means as in [209] or co-clustering techniques as in [88] on this augmented feature set.

For the syntactic approaches, the trick is achieved through the score or cost functions by incorporating contexts into them. For example, one can reward or penalize the substitution of activities at some position depending on how close their execution times are. Equation (4.1) can be modified as

$$ged(\mathbf{s},\mathbf{t}) = \min \begin{cases} \mathcal{C}(\mathbf{s}(|\mathbf{s}|),\mathbf{t}(|\mathbf{t}|)) + \psi(\mathbf{s}(|\mathbf{s}|),\mathbf{t}(|\mathbf{t}|)) + ged(\mathbf{s}^{|\mathbf{s}|-1},\mathbf{t}^{|\mathbf{t}|-1}), \\ \mathcal{C}(\mathbf{s}(|\mathbf{s}|),-) + ged(\mathbf{s}^{|\mathbf{s}|-1},\mathbf{t}), \\ \mathcal{C}(-,\mathbf{t}(|\mathbf{t}|)) + ged(\mathbf{s},\mathbf{t}^{|\mathbf{t}|-1}) \end{cases} \tag{4.5}$$

where the cost function $\psi : \mathcal{E} \times \mathcal{E} \to \mathbb{R}$ defined over the events in an event log rewards the substitution of two events, $e_1, e_2 \in \mathcal{E}$, if their activity execution times are closer to each other. The larger the difference between their execution times, the larger is the cost (penalty), thus discouraging their substitution. Extending syntactic approaches with additional perspectives is an interesting topic of research that deserves further exploration.

4.10 Conclusions

In this chapter, we advocated the use of clustering to deal with heterogeneity and noise. Process mining results can be improved by partitioning the event log into homogenous sets of cases and analyzing each subset independently. We argued that considering contexts of execution is important in deriving good clusters. We extended the current work on trace clustering by defining process-centric feature sets and syntactic methods based on edit distance. In order to tackle the sensitivity of the cost function (of edit operations) in the generic edit distance framework, we proposed an algorithm that automatically derives the scores for edit operations. Experiments show that the scores derived using this approach are effective. The proposed methods outperform contemporary approaches to trace clustering. Furthermore, we showed that the process-centric feature sets, especially the maximal repeat alphabet, are not only effective but also computationally efficient. This makes our approach amenable for large scale event logs.

Chapter 5
Concept Drift

In this chapter, we look at techniques for dealing with process changes. Operational processes need to change to adapt to changing circumstances, e.g., new legislation, extreme variations in supply and demand, seasonal effects, etc. For example, a new medical regulation might have an influence on the way a particular procedure is applied by the physicians using a hospital information system. In case of a disaster, hospitals and banks may change their operating procedures, etc. Furthermore, in today's dynamic marketplace, it is increasingly necessary for enterprises to streamline their processes so as to reduce costs and to improve performance. The challenge to keep a step ahead of competitors forces organizations to adapt their processes continuously [123]. The power of modern information systems is a major impetus for designing flexible business processes and facilitating business process change or reengineering [90]. For example, process-aware information systems (PAIS) have been extended to be able to flexibly adapt to changes in the process. State-of-the-art Workflow Management (WFM) and BPM systems provide such flexibility, e.g., one can easily release a new version of a process. Moreover, in processes not driven by WFM/BPM systems (such as the usage of medical systems) there is even more flexibility as processes are controlled by people rather than information systems.

Although flexibility and change have been studied in-depth in the context of WFM and BPM systems, *existing process mining techniques assume processes to be in steady state*. For example, when discovering a process model from event logs, it is assumed that the process at the beginning of the recorded period is the same as the process at the end of the recorded period. However, this is a very unrealistic assumption to make as process changes could have taken place. An idealistic requirement is to record the sequence of process changes performed, in the form of a change log, so that event logs can be analyzed appropriately [95]. However, such a change log will not be available in most cases. As a result, process changes manifest themselves only *latently* in the event logs (in the way which activities are executed when, how, and by whom). Analyzing such changes is of the utmost importance to get an accurate insight on process executions at any instant of time.

In the data mining and machine learning communities, such second-order dynamics are referred to as *concept drift*. Concept drift refers to changes in the target variable(s)/concept induced by contextual shifts over time [267]. We use the term of concept drift to refer to changes in processes as well. When dealing with concept drifts in process mining, the following three main challenges emerge:

1. *Change (Point) Detection:* The first and most fundamental problem is to detect concept drift in processes, i.e., to detect that a process change has taken place. If so, the next step is to identify the time periods at which changes have taken place.
2. *Change Localization and Characterization:* Once a point of change has been identified, the next step is to characterize the nature of change, and identify the region(s) of change (localization) in a process. Uncovering the nature of change is a challenging problem that involves both the identification of change perspective (e.g., control-flow, data, resource, sudden, gradual, etc.) and the identification of the exact change itself.
3. *Unravel Process Evolution:* Having identified, localized, and characterized the changes, it is necessary to put all of these in perspective. There is a need for techniques/tools that exploit and relate these discoveries. Unraveling the evolution of a process should result in the discovery of the change process describing the second order dynamics.

One can differentiate between two broad classes of dealing with concept drifts when analyzing event logs.

- *offline analysis:* this refers to the scenario where the presence of changes or the occurrence of drifts need not be uncovered in real-time. This is appropriate in cases where the detection of changes is mostly used in postmortem analysis, the results of which can be considered when designing/improving processes for later deployment.
- *online analysis:* this refers to the scenario where the presence of changes or the occurrence of drifts need to be discovered in near real-time. This is appropriate in cases where an organization would be more interested in knowing a change in behavior of their customers or a change in demand *as and when* it is happening. Such near real-time triggers (alarms) will enable organizations to take quick remedial actions and avoid any repercussions.

In this chapter, we focus on two of the challenges, viz., change (point) detection and change localization and characterization in an offline setting. We propose features and techniques to detect changes (drifts), change points, and change localization in event logs from a control-flow perspective. Detection of such change points enables the selection of cases and putting the analysis results in perspective to process variants.

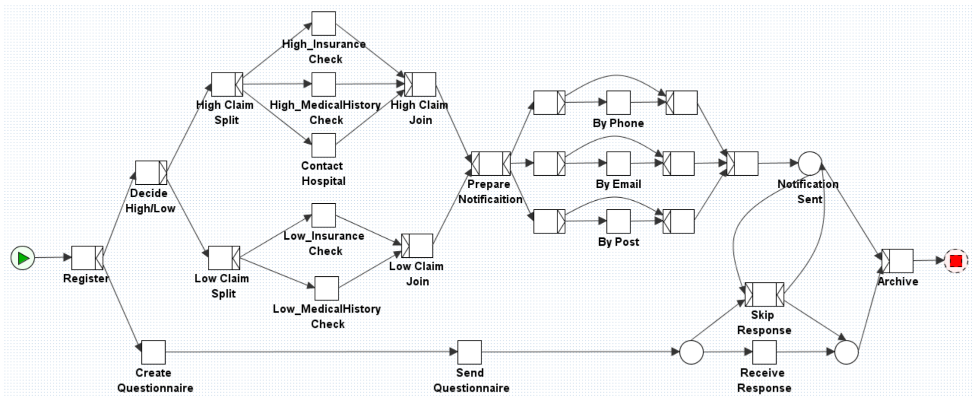
The remainder of this chapter is organized as follows. Section 5.1 presents a running example to explain the concepts presented in this chapter. Related work is presented in Section 5.2. Section 5.3 describes the various aspects and nature of change. Section 5.4 introduces various features and techniques for detecting drifts in event logs. Section 5.6 describes the effectiveness of the features and techniques proposed in this chapter in discovering change points and localization of changes. In Section 5.7, we provide an outlook on some of the open research questions and directions for future research in this area. Finally, Section 5.8 concludes the chapter.

5.1 Running Example

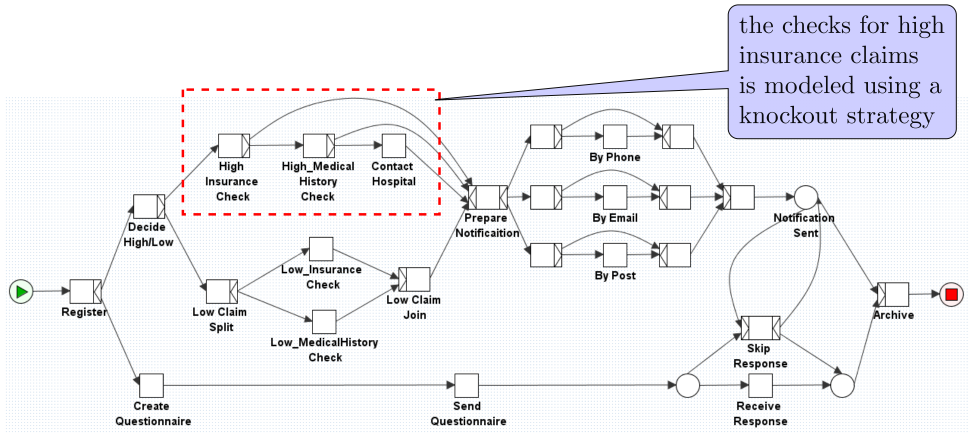
The digital copier example that we described in Chapter 3 captures the workflow of a copier for copy/scan and print job requests. The workflow of the copier can change over time, e.g., new features may be added, existing features may be optimized for performance, etc. However, the activities involved in the workflow are all automated tasks executed by the various components in the copier. In order to make the topic of concept drift more interesting, we consider a different process, viz., an insurance claim process which contains tasks executed by people, as our running example. We revisit the digital copier example in Section 5.6.

The insurance claim process that we use as our running example corresponds to the handling of health insurance claims in a travel agency. Upon registration of a claim, a general questionnaire is sent to the claimant. In parallel, a registered claim is classified as high or low. For low claims, two independent tasks, viz., check insurance and check medical history need to be executed. For high claims, three tasks need to be executed, viz., check insurance, check medical history, and contact doctor/hospital for verification. If one of the checks shows that the claim is not valid, then the claim is rejected; otherwise, it is accepted. A cheque and acceptance decision letter is prepared in cases where a claim is accepted while a rejection decision letter is created for rejected claims. In both cases, a notification is sent to the claimant. Three modes of notification are supported, viz., by email, by telephone (fax), and by postal mail. The case should be archived upon notifying the claimant. This can be done with or without the response for the questionnaire. However, the decision of ignoring the questionnaire can only be made after a notification is sent. The case is closed upon completion of archiving task.

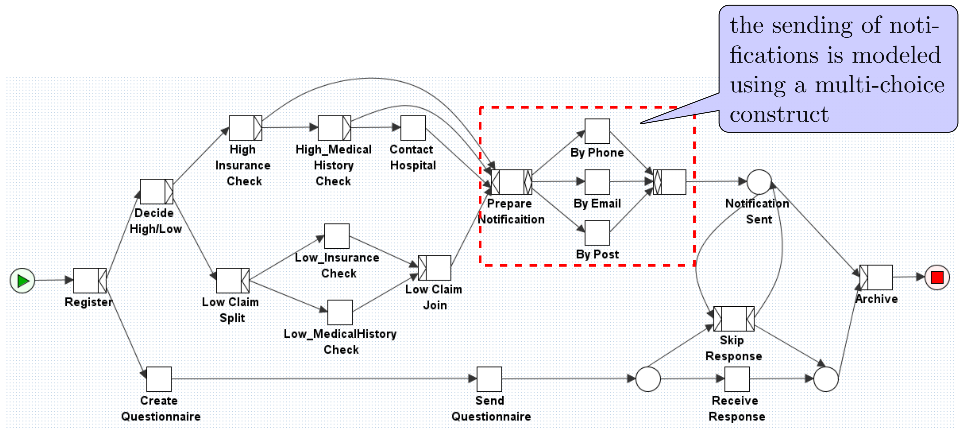
Figure 5.1 depicts five variants of this process represented in YAWL [224] notation. The dashed rectangles indicate regions where a change has been done in the process model with respect to its previous variant. The changes can have various reasons. For example, in Figure 5.1(a), the different checks for high insurance claims are modeled using a parallel (AND) construct. However, a claim can be rejected if any one of the checks fail. In such cases, the time and resources spent on other checks go waste. To optimize this process, the agency can decide to enforce an order on these checks and proceed on checks only if the previous check results are positive. In other words, the process is modified with a *knockout* strategy [216] adopted for the process fragment involving the different checks for high insurance claims as depicted in Figure 5.1(b). As another example, the OR-construct pertaining to the sending of notification to claimants in Figure 5.1(c) has been modified to an exclusive-or (XOR) construct in Figure 5.1(d). The organization could have taken a decision to reduce their workforce as a cost-cutting measure. Due to availability of limited resources, they would like to minimize the redundancy of sending the notification through different modes of communication and restrict it to only one of the modes. Considering an event log containing cases that belong to such a mix of process variants, the objective of change point detection is to detect when the processes have changed as illustrated in Figure 5.2.



(a) M1

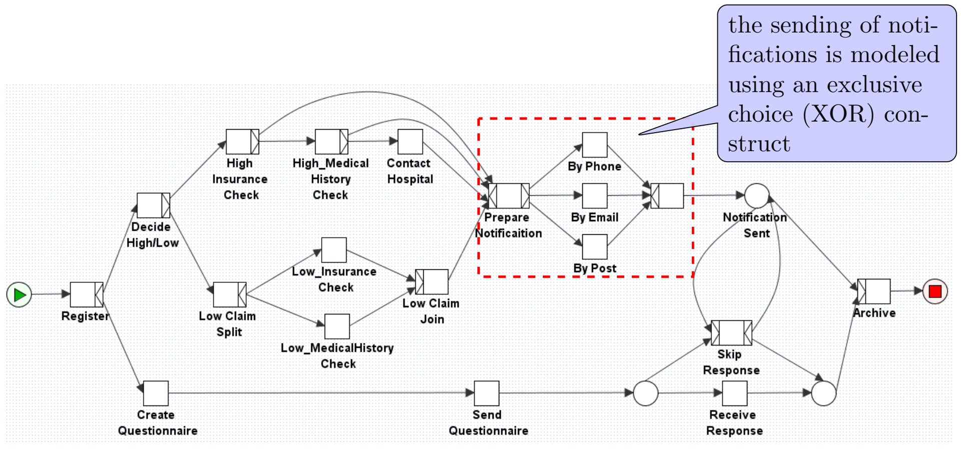


(b) M2

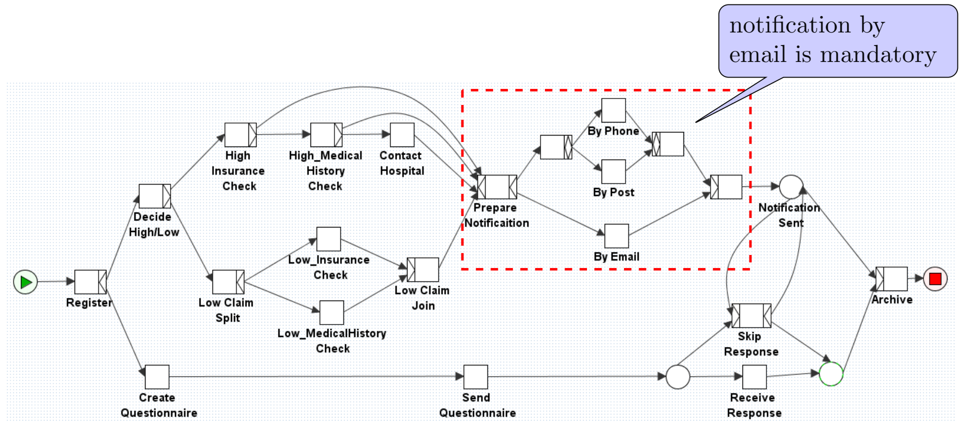


(c) M3

Figure 5.1: Variants 1-3 (of 5) of an insurance claim process of a travel agency represented in YAWL notation. The dashed rectangles indicate the regions of change from its previous model.



(a) M4



(b) M5

Figure 5.1: Variants 4-5 (of 5) of an insurance claim process (cont.).

5.2 Related Work

Over the last two decades many researchers have been working on process flexibility, e.g., making workflow systems adaptive. In [156, 261] collections of typical change patterns are described. In [181, 201] extensive taxonomies of the various flexibility approaches and mechanisms are provided. Ploesser et al. [174] have classified business process changes into three broad categories, viz., sudden, anticipatory, and evolutionary. This classification is used in this chapter, but now in the context of event logs.

Despite the many publications on flexibility, most process mining techniques assume a process to be in steady state. A notable exception is the approach by Günther et

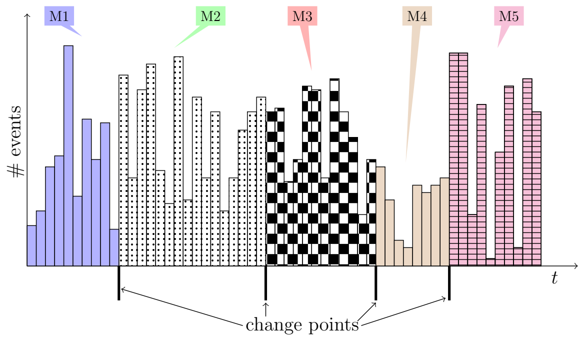


Figure 5.2: An event log containing traces from different process variants along with the points of change.

al. [95]. This approach uses process mining to provide an aggregated overview of all changes that have happened so far. However, this approach assumes that change logs are available, i.e., modifications of the workflow model are recorded. At this point in time very few information systems provide such change logs. Therefore, *this chapter focuses on concept drift in process mining assuming only an event log as input.*

The topic of concept drift is well-studied in various branches of the data mining and machine learning community over the last decade. Concept drift research primarily has been focusing on two directions: (a) how to detect drifts (changes) online (e.g., [16, 78, 162, 187, 248]); (b) how to keep predictive models up to date (e.g., [64, 132, 151, 267]). Concept drift has been studied in both supervised and unsupervised settings and has been shown to be important in many applications [168, 199, 213, 252, 253, 257]. However, the problem of concept drift has not been studied in the process mining community. Unlike in data mining and machine learning, where concept drift focusses on changes in simple structures such as variables, concept drift in process mining deals with changes to complex artifacts such as process models describing concurrency, choices, loops, cancelation, etc. Although experiences from data mining and machine learning can be used to investigate concept drift in process mining, existing techniques cannot be used due to the complexity of process models and the nature of process change.

5.3 Change in Business Processes

In this section, we discuss on the various aspects of process change.

5.3.1 Perspectives of Change

There are three important perspectives in the context of business processes, viz., control-flow, data, and resource. One or more of these perspectives may change over time.

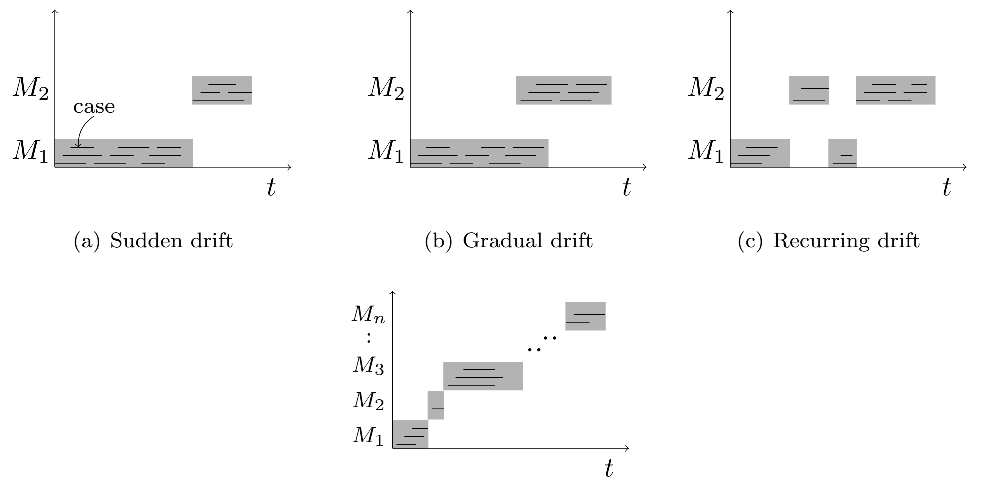
- *Control-flow/Behavioral Perspective:* This class of changes deals with the behavioral and structural changes in a process model. Just like the design patterns in software engineering, there exist *change patterns* capturing the common control-flow changes [261]. Control-flow changes can be classified into operations such as insertion, deletion, substitution, and reordering of process fragments. For example, an organization which used to collect a fee after processing and acceptance of an application can now change their process to enforce payment of that fee before processing an application. Here, the *reordering* change pattern had been applied on the payment and application processing process fragments. As another example, with the addition of new product offerings, a *choice* construct is *inserted* into the product development process of an organization. In the context of PAISs, various control-flow change patterns have been proposed in [156, 261]. Most of these control-flow change patterns are applicable to traditional information/workflow systems as well.

Sometimes, the control-flow structure of a process model can remain intact but the behavioral aspects of a model change. For example, consider an insurance agency that classifies claims as “high” or “low” depending on the amount claimed. An insurance claim of €1000 which would have been classified as high last year is categorized as a low insurance claim this year due to the organization’s decision to increase the claim limit. The structure of the process remains intact but the routing of cases changes.

- *Data Perspective:* This class of changes refer to the changes in the production and consumption of data and the effect of data on the routing of cases. For example, it may no longer be required to have a particular document when approving a claim.
- *Resource Perspective:* This class deals with the changes in resources, their roles, and organizational structure, and their influence on the execution of a process. For example, there could have been a change pertaining to who executes an activity. Roles may change and people may change roles. As another example, certain execution paths in a process could be enabled (disabled) upon the availability (non-availability) of resources. Furthermore, resources tend to work in a particular manner and such working patterns may change over time, e.g., a resource can have a tendency of executing a set of parallel activities in a specific sequential order. Such working patterns could be more prominent when a limited number of resources are available; the addition of new resources can remove this bias.

5.3.2 Nature of Drifts

Based on the duration for which a change is active, one can classify changes into *momentary* and *permanent*. Momentary changes are short-lived and affect only a very few cases while permanent changes are persistent and stay for a while [201]. Momentary changes correspond to the notion of outliers in statistics. In this chapter, we consider only permanent changes. Changes are perceived to induce a drift in the concept (process behavior). As depicted in Figure 5.3, we identify four classes of drifts.



(a) Sudden drift (b) Gradual drift (c) Recurring drift

(d) Incremental drift

Figure 5.3: Different types of drifts. X-axis indicates time and Y-axis indicates process variants. Shaded rectangles depict process instances.

- *Sudden Drift:* This corresponds to a substitution of an existing process M_1 with a new process M_2 as depicted in Figure 5.3(a). M_1 ceases to exist from the moment of substitution. In other words, all cases (process instances) from the instant of substitution emanate from M_2. This class of drifts is typically seen in scenarios such as emergencies, crisis situations, and change of law. As an example, a new regulation by the finance ministry of India mandates all banks to procure and report the customer's Personal Account Number (PAN) in their transactions.

- *Gradual Drift:* This refers to the scenario as depicted in Figure 5.3(b) where a current process M_1 is replaced with a new process M_2. Unlike the sudden drift, here both processes coexist for some time with M_1 discontinued gradually. For example, a supply chain organization might introduce a new delivery process. However, this process is applicable only for orders taken henceforth. All previous orders still have to follow the former delivery process.

- *Recurring Drift:* This corresponds to the scenario where a set of processes reappear after some time (substituted back and forth) as depicted in Figure 5.3(c). It is quite natural to see such a phenomenon with processes having a seasonal influence. For example, a travel agency might deploy a different process to attract customers during Christmas period. The recurrence of processes may be periodic or non-periodic. An example of a non-periodic recurrence is the deployment of a process subject to market conditions. The point of deployment and the duration of deployment are both dependent on external factors (here, the market conditions).

- *Incremental Drift:* This refers to the scenario where a substitution of process M_1 with M_N is done via smaller incremental changes as depicted in Figure 5.3(d). This class of drifts is more pronounced in organizations adopting agile business process management methodology and in processes undergoing quality improvements (most Total Quality Management (TQM) initiatives are examples of incremental change [103]).

Recurring and incremental drifts in Figure 5.3 are depicted as drifts occurring with a sudden phenomenon. However, though uncommon, they can also occur in a gradual phenomenon. In the remainder, we propose approaches to detect potential control-flow changes in a process manifested as sudden drifts over a period of time by analyzing its event log. Detecting drifts in data and resource perspectives and in the contexts of gradual drifts is beyond the scope of this thesis.

5.4 Approaches to Detecting Drifts in Event Logs

In this section, we propose techniques that enable the detection of changes and the points of change by analyzing event logs.

One can consider an event log $\mathcal{L}$ as a time series of traces (traces ordered based on the timestamp of the first event). Figure 5.4 depicts such a perspective on an event log along with change points in the sudden drift scenario. The basic premise in handling concept drifts is that the *characteristics of the traces before the change point* differ from the *characteristics of the traces after the change point.* The problem of change point detection is then to identify the points in time where the process has changed, if any. Change point detection involves two primary steps:

1. capturing the characteristics of the traces, and
2. identifying when the characteristics change

We refer to the former step as *feature extraction* and the latter step as *drift detection.* The characteristics of the traces can either be defined for each trace separately or can be done at a sub-log level. An event log can be split into sub-logs of s traces ($s \in \mathbb{N}$ is the split size). One can consider either overlapping or non-overlapping windows when creating such sub-logs. Figure 5.4 depicts the scenario where two subsequent sub-logs do not overlap. In this case, we have $k = \lceil \frac{n}{s} \rceil$ sub-logs for an event log of n traces.

As mentioned earlier, dealing with concept drifts involves two primary steps. First, we need to capture the characteristics of traces; we propose a few feature sets that address this in Section 5.4.1. Second, we need to identify when these characteristics change; we look at techniques that address this in Section 5.4.2.

5.4.1 Features Capturing the Manifestation of Activity Relationships

Event logs are *characterized* by the relationships between activities. Dependencies between activities in an event log can be captured and expressed using the *follows* (or

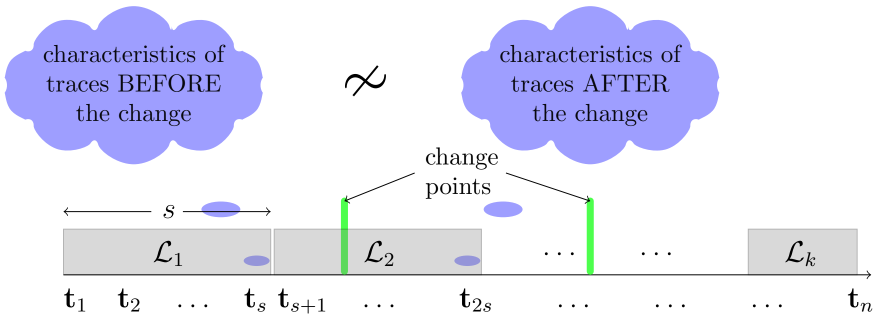

characteristics of traces BEFORE the change
$\not\propto$
characteristics of traces AFTER the change
change points
s
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_k$
$\mathbf{t}_1$ $\mathbf{t}_2$ ... $\mathbf{t}_s$ $\mathbf{t}_{s+1}$ ... $\mathbf{t}_{2s}$ ... ... $\mathbf{t}_n$

Figure 5.4: An event log visualized as a time series of traces along with change points. The basic premise of change (point) detection is that characteristic differences exist in the traces before and after the change.

precedes) relationship, also referred to as *causal footprints*. For any pair of activities, a, b ∈ $\mathcal{A}$, one can determine whether they exhibit either *always*, *never*, or *sometimes* follows/precedes relationship. If b follows a in all the traces in an event log, then we say that b *always follows* a; if b follows a only in some subset of the traces or in none of the traces, then we say that b *sometimes follows* a, and b *never follows* a respectively. Consider an event log $\mathcal{L}$ = [acaebfh, ahijebd, aeghijk] containing three traces defined over $\mathcal{A}$ ={a, b, c, d, e, f, g, h, i, j, k}. The following relations hold in $\mathcal{L}$: e *always follows* a, e *never follows* b, and b *sometimes follows* a. Figure 5.5(a) depicts the relationship between every pair of activities in $\mathcal{A}$. The value in a cell (i, j) is either A, S, or N corresponding to the relation whether the activity represented by column j always, sometimes, or never follows the activity represented by row i respectively.

The variants of *precedes* relation can be defined along similar lines. The *follows/precedes* relationship is rich enough to reveal many control flow changes in a process. We exploit this relationship and define various features for change detection.

We distinguish between two classes of features: (i) *global* features and (ii) *local* features. Global features are defined over an event log while local features can be defined at a trace level. Based on the follows (precedes) relation, we propose two global features, viz., Relation Type Count and Relation Entropy, and two local features, viz., Window Count and J-measure. These features are defined as follows:

- *Relation Type Count (RC):* The relation type count with respect to the follows (precedes) relation is a function, $f_{\text{RC}} : \mathcal{A} \to \mathbb{N}_0^3$, defined over the set of activities $\mathcal{A}$. f_{RC} of an activity, x ∈ $\mathcal{A}$, with respect to the follows (precedes) relation over an event log $\mathcal{L}$ is the triple $\langle c_A, c_S, c_N \rangle$ where c_A, c_S, and c_N are the number of activities in $\mathcal{A}$ that always, sometimes, and never follows (precedes) x, respectively, in the event log $\mathcal{L}$. For the event log $\mathcal{L}$ mentioned above, f_{RC}(a) = $\langle 2, 9, 0 \rangle$ since e and h always follows a while all other activities in $\mathcal{A} \setminus$ {e, h} sometimes follows a. f_{RC}(i) = $\langle 1, 4, 6 \rangle$ since only j always follows i; b, d, e, and k sometimes follows i while a, c, f, g, h, and i never follows

	a	b	c	d	e	f	g	h	i	j	k	f_{RC}	f_{RE}
a	S	S	S	S	A	S	S	A	S	S	S	$\langle 2,9,0 \rangle$	0.684
b	N	N	N	S	N	S	N	S	N	N	N	$\langle 0,3,8 \rangle$	0.845
c	A	A	N	N	A	A	N	A	N	N	N	$\langle 5,0,6 \rangle$	0.994
d	N	N	N	N	N	N	N	N	N	N	N	$\langle 0,0,11 \rangle$	0.000
e	N	S	N	S	N	S	S	S	S	S	S	$\langle 0,8,3 \rangle$	0.845
f	N	N	N	N	N	N	N	A	N	N	N	$\langle 1,0,10 \rangle$	0.440
g	N	N	N	N	N	N	N	A	A	A	A	$\langle 4,0,7 \rangle$	0.946
h	N	S	N	S	S	N	N	N	S	S	S	$\langle 0,6,5 \rangle$	0.994
i	N	S	N	S	S	N	N	N	N	A	S	$\langle 1,4,6 \rangle$	1.322
j	N	S	N	S	S	N	N	N	N	N	S	$\langle 0,4,7 \rangle$	0.946
k	N	N	N	N	N	N	N	N	N	N	N	$\langle 0,0,11 \rangle$	0.000

(a) (b) (c)

Figure 5.5: (a) Causal footprint matrix for all activity pairs – A: Always follows, N: Never follows and S: Sometimes follows (b) Relation type count for all activities, and (c) Relation entropy for all activities.

i. Figure 5.5(b) depicts the relation type counts for all the activities in $\mathcal{A}$ (the value in a row corresponds to the relation type counts of the activity represented by that row in Figure 5.5(a)).

For an event log containing $|\mathcal{A}|$ activities, this results in a feature vector of dimension $3 \times |\mathcal{A}|$ (if either the follows or the precedes relation is considered) or $2 \times 3 \times |\mathcal{A}|$ (if both the follows and the precedes relations are considered).

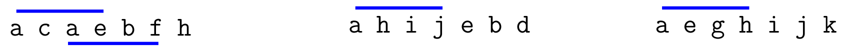
- *Relation Entropy (RE):* The relation entropy with respect to the follows (precedes) relation is a function, $f_{\mathrm{RE}} : \mathcal{A} \to \mathbb{R}_0^+$, defined over the set of activities. f_{RE} of an activity, $\mathtt{x} \in \mathcal{A}$ with respect to the follows (precedes) relation is the entropy of the relation type count metric. In other words, $f_{\mathrm{RE}}(\mathtt{x}) = -p_A \log_2(p_A) - p_S \log_2(p_S) - p_N \log_2(p_N)$ where $p_A = c_A/|\mathcal{A}|$, $p_S = c_S/|\mathcal{A}|$, and $p_N = c_N/|\mathcal{A}|$ and $\langle c_A, c_S, c_N \rangle$ is the relation count value of the activity $\mathtt{x}$.

  For the above example event log $\mathcal{L}$, $f_{\mathrm{RE}}(\mathtt{a}) = 0.684$ (corresponding to $f_{\mathrm{RC}}(\mathtt{a}) = \langle 2,9,0 \rangle$) and $f_{\mathrm{RE}}(\mathtt{i}) = 1.322$ (corresponding to $f_{\mathrm{RC}}(\mathtt{i}) = \langle 1,4,6 \rangle$). Figure 5.5(c) depicts the relation entropy for all the activities in $\mathcal{A}$ (the value in a row corresponds to the relation entropy of the activity represented by that row in Figure 5.5(a)).

For an event log containing $|\mathcal{A}|$ activities, this results in a feature vector of dimension $|\mathcal{A}|$ or $2 \times |\mathcal{A}|$ depending on whether either or both of the follows/precedes relations are considered.

- *Window Count (WC):* Given a window of size $l \in \mathbb{N}$, the window count with respect to follows (precedes) relation is a function, $f^l_{\mathrm{WC}} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{N}_0$, defined over the set of activity pairs. Given a trace $\mathbf{t}$ and a window of size l, let $S^l(\mathtt{a})$ be the bag of all subsequences $\mathtt{t}(i, i+l-1)$, such that $\mathtt{t}(i) = \mathtt{a}$[1]. Let $\mathcal{F}^l(\mathtt{a},\mathtt{b}) = [\mathtt{s} \in S^l(\mathtt{a}) \mid \exists_{1<k\leq|\mathtt{s}|}\ \mathtt{s}(k) = \mathtt{b}]$, i.e., the bag of subsequences in $\mathbf{t}$ starting with $\mathtt{a}$ and followed by $\mathtt{b}$ within a window of length l. The window count of the relation $\mathtt{b}$ *follows* $\mathtt{a}$, $f^l_{\mathrm{WC}}(\mathtt{a},\mathtt{b}) = |\mathcal{F}^l(\mathtt{a},\mathtt{b})|$.

Figure 5.6 depicts the window count values for the relation $\mathtt{b}$ *follows* $\mathtt{a}$ in the event log $\mathcal{L}$ using a window of length 4.



$$S^4(\mathtt{a}) = [\mathtt{acae}, \mathtt{aebf}] \quad S^4(\mathtt{a}) = [\mathtt{ahij}] \quad S^4(\mathtt{a}) = [\mathtt{aegh}]$$
$$\mathcal{F}^4(\mathtt{a},\mathtt{b}) = [\mathtt{aebf}] \quad \mathcal{F}^4(\mathtt{a},\mathtt{b}) = [\,] \quad \mathcal{F}^4(\mathtt{a},\mathtt{b}) = [\,]$$
$$f^4_{\mathrm{WC}}(\mathtt{a},\mathtt{b}) = 1 \quad f^4_{\mathrm{WC}}(\mathtt{a},\mathtt{b}) = 0 \quad f^4_{\mathrm{WC}}(\mathtt{a},\mathtt{b}) = 0$$

Figure 5.6: Window count values for the relation b follows a for the different traces in the event log.

- *J-Measure:* Smyth and Goodman [204] have proposed a metric called J-measure based on [18] to quantify the information content (goodness) of a rule. We adopt this metric as a feature to characterize the significance of relationship between activities. The basis lies in the fact that one can consider the relation $\mathtt{b}$ follows $\mathtt{a}$ as a rule: "if activity $\mathtt{a}$ occurs, then activity $\mathtt{b}$ will probably occur". The J-measure with respect to follows (precedes) relation is a function $f^l_{\mathrm{J}} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}^+$ defined over the set of activity pairs and a given window of length $l \in \mathbb{N}$.

  Let $p(\mathtt{a})$ and $p(\mathtt{b})$ denote the probability of occurrence of activities $\mathtt{a}$ and $\mathtt{b}$ respectively in a trace $\mathbf{t}$. Let $p^l(\mathtt{a},\mathtt{b})$ denote the probability that $\mathtt{b}$ *follows* $\mathtt{a}$ within a window of length l, i.e., $p^l(\mathtt{a},\mathtt{b}) = |\mathcal{F}^l(\mathtt{a},\mathtt{b})|/|S^l(\mathtt{a})|$. Then the J-measure for a window of length l is defined as $f^l_J(\mathtt{a},\mathtt{b}) = p(\mathtt{a})\mathrm{CE}^l(\mathtt{a},\mathtt{b})$ where $\mathrm{CE}^l(\mathtt{a},\mathtt{b})$ denotes the cross-entropy of $\mathtt{a}$ and $\mathtt{b}$ ($\mathtt{b}$ follows $\mathtt{a}$ within a window of length l) and is defined as[2]

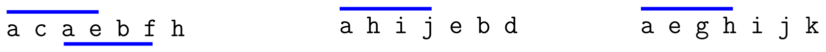
$$\mathrm{CE}^l(\mathtt{a},\mathtt{b}) = p^l(\mathtt{a},\mathtt{b}) \log_2\left(\frac{p^l(\mathtt{a},\mathtt{b})}{p(\mathtt{b})}\right) + (1 - p^l(\mathtt{a},\mathtt{b})) \log_2\left(\frac{1 - p^l(\mathtt{a},\mathtt{b})}{1 - p(\mathtt{b})}\right)$$

  The J-measure of a relation, $\mathtt{b}$ follows $\mathtt{a}$, captures the dissimilarity between the apriori and aposteriori beliefs about $\mathtt{b}$. In other words, it measures the

[1]if $i + l - 1 > |\mathbf{t}|$, then $\mathbf{t}(i, i + l - 1) = \mathbf{t}(i, |\mathbf{t}|)$, i.e., the suffix of the trace $\mathbf{t}$ starting at i.
[2]$\log_2(0/x)$ and $\log_2(x/0)$ for any $x \in \mathbb{R}^+_0$ is taken as 0.

difference between the priori distribution of `b` (i.e., probability that `b` occurs in a trace and the probability that `b` does not occur), and the posteriori distribution of `b` (i.e., probability that `b` occurs in a trace given that `a` occurred and the probability that `b` does not occur in a trace given that `a` occurred).

Figure 5.7 depicts the J-measure for the relation `b` *follows* `a` in the event log $\mathcal{L}$ using a window of length 4.



$$\begin{aligned} p(\mathtt{a}) &= 2/7 = 0.286 \\ p(\mathtt{b}) &= 1/7 = 0.143 \\ p^4(\mathtt{a},\mathtt{b}) &= 0.5 \\ \mathrm{CE}^4(\mathtt{a},\mathtt{b}) &= 0.514 \\ f_J^4(\mathtt{a},\mathtt{b}) &= 0.147 \end{aligned} \qquad \begin{aligned} p(\mathtt{a}) &= 1/7 = 0.143 \\ p(\mathtt{b}) &= 1/7 = 0.143 \\ p^4(\mathtt{a},\mathtt{b}) &= 0 \\ \mathrm{CE}^4(\mathtt{a},\mathtt{b}) &= 0.514 \\ f_J^4(\mathtt{a},\mathtt{b}) &= 0.032 \end{aligned} \qquad \begin{aligned} p(\mathtt{a}) &= 1/7 = 0.143 \\ p(\mathtt{b}) &= 0 \\ p^4(\mathtt{a},\mathtt{b}) &= 0 \\ \mathrm{CE}^4(\mathtt{a},\mathtt{b}) &= 0 \\ f_J^4(\mathtt{a},\mathtt{b}) &= 0 \end{aligned}$$

Figure 5.7: J-measure values for the relation `b` follows `a` for the different traces in the event log.

Having defined the features, we next look at the second step in change point detection, i.e., drift detection.

5.4.2 Statistical Hypothesis Tests to Detect Drifts

An event log can be transformed into a data set $\mathcal{D}$ by choosing one of the feature sets defined in the previous section. The dataset $\mathcal{D}$ of feature values can be considered as a time series as depicted in Figure 5.8. Each $\mathbf{d}_i \in \mathcal{D}$ corresponds to the feature value(s) for a trace (or sub-log) and can be a scalar or a vector (depending on the choice of feature)[3]. Comparing with Figure 5.4, $m = n$ or $m = k$ depending on whether the feature values are computed for each trace or for each sub-log respectively. As mentioned earlier, we expect a characteristic difference in the manifestation of feature values in the traces (sub-logs) before and after the change points with the difference being more pronounced at the boundaries. In order to detect this, *we can consider a series of successive populations of values (of size w) and investigate if there is a significant difference between the two populations.* The premise is that differences are expected to be perceived at change points provided appropriate characteristics of the change are captured as features. A moving window of size w is used to generate the populations. Figure 5.8 depicts a scenario where two populations $P_1 = \langle \mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_w \rangle$ and $P_2 = \langle \mathbf{d}_{w+1}, \mathbf{d}_{w+2}, \ldots, \mathbf{d}_{2w} \rangle$ of size w are considered. In the next iteration, the populations correspond to $P_1 = \langle \mathbf{d}_2, \mathbf{d}_3, \ldots, \mathbf{d}_{w+1} \rangle$ and $P_2 = \langle \mathbf{d}_{w+2}, \mathbf{d}_{w+3}, \ldots, \mathbf{d}_{2w+1} \rangle$. Given a data set of m values, the number of population pairs (iterations) will be $m - 2w + 1$.

We propose the use of *statistical hypothesis testing* to discover these change points.

[3]The RE, WC, and J-measure feature sets proposed in Section 5.4.1 generate univariate (scalar) and multi-variate (vector) data depending on whether we consider an individual activity/activity-pair or a set of activities/activity pairs respectively. The RC feature set always generates multi-variate data.

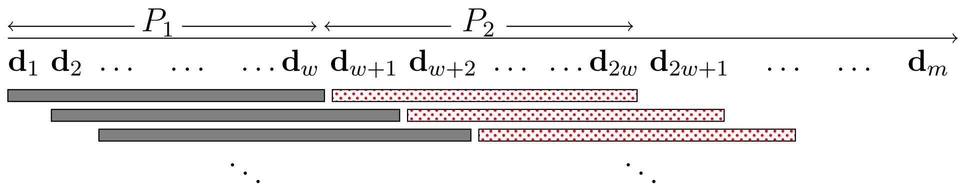

P_1
P_2
$\mathbf{d}_1$ $\mathbf{d}_2$ … … … $\mathbf{d}_w$ $\mathbf{d}_{w+1}$ $\mathbf{d}_{w+2}$ … … $\mathbf{d}_{2w}$ $\mathbf{d}_{2w+1}$ … … $\mathbf{d}_m$

Figure 5.8: Basic idea of detecting drifts using hypothesis tests. The data set of feature values is considered as a time series for hypothesis tests. P_1 and P_2 are two populations of size w.

Hypothesis testing is a procedure in which a *hypothesis* is evaluated on a sample data. One of the important uses of hypothesis testing is to evaluate and compare groups of data. Numerous varieties of hypothesis tests exist [120, 202]. The choice of a particular test is largely dependent on the nature of the data and the objectives of an experiment. For example, hypothesis tests can be classified into *parametric* and *non-parametric* tests. Parametric tests assume that the data have a particular distribution, e.g., normal, while the non-parametric tests do not make any assumption with regards to the data distribution. Since we do not know the apriori distribution of the feature values in an event log, we consider only the non-parameteric tests. Another perspective of classification is based on the number of samples (populations) on which the hypothesis is defined. One can classify the hypothesis tests into (i) *one-sample*, (ii) *two-sample*, and (iii) *multi-sample* tests. Since we need to analyze two populations for detecting drifts, we are interested in two-sample hypothesis tests. Another classification of hypothesis tests is concerned with the dimensionality of each data element in a sample. Tests dealing with scalar data elements are called as *univariate tests* while those dealing with vector data elements are called as *multi-variate tests*. If only a particular activity or activity pair is considered, then every data item $\mathbf{d}_i \in \mathcal{D}$ is a scalar value corresponding to the trace/sub-log i. However, if we consider sets of activities or activity pairs, then each data item is a vector. Therefore, we need to consider both univariate and multi-variate hypothesis tests.

We will use the univariate two-sample *Kolmogorov-Smirnov* test (*KS* test) and *Mann-Whitney U* test (*MW* test) as hypothesis tests for univariate data, and the two sample *Hotelling* T^2 test for multivariate data. The *KS* test evaluates the hypothesis "Do the two independent samples represent two different cumulative frequency distributions?" while the *MW* test evaluates the hypothesis "Do the two independent samples have different distributions with respect to the rank-ordering of the values?". The multi-variate Hotelling T^2 test is a generalization of the t-test and evaluates the hypothesis "Do the two samples have the same mean pattern?". All of these tests yield a *significance probability* assessing the validity of the hypothesis on the samples. The MW test is computationally more expensive than the KS test as the data has to be sorted (to estimate the rank ordering). We refer the reader to [202] for a classic introduction to various hypothesis tests.

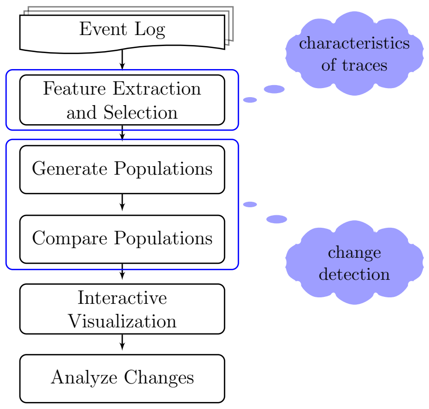

Event Log
Feature Extraction and Selection
Generate Populations
Compare Populations
Interactive Visualization
Analyze Changes
characteristics of traces
change detection

Figure 5.9: Framework for handling concept drifts in process mining.

5.5 Framework

We propose the framework depicted in Figure 5.9 for handling concept drifts in process mining. The framework identifies the following steps:

- *Feature Extraction and Selection:* This step pertains to defining the characteristics of the traces in an event log. In this chapter, we have defined four features that characterize the control-flow perspective of process instances in an event log. Depending on the focus of analysis, one may define additional features, e.g., if we are interested in analyzing changes in organizational/resource perspective, we may consider features derived from social networks as a means of characterizing the event log. In addition to feature extraction, this step also involves feature selection. Feature selection is important when the number of features extracted is large. One may consider dimensionality reduction techniques [71, 99] such as PCA [119] and random projection [17] to deal with high-dimensionality.
- *Generate Populations:* An event log can be transformed into a data stream based on the features selected in the previous step. This step deals with defining the sample populations for studying the changes in the characteristics of traces. Different criteria/scenarios may be considered for generating these populations from the data stream. In Section 5.4.2, we have considered non-overlapping, continuous, and fixed-size windows for defining the populations. One may also consider, for example, non-continuous windows (there is a gap between two populations), adaptive windows (windows can be of different lengths) [16], etc., which are more appropriate for dealing with gradual and recurring drifts.
- *Compare Populations:* Once the sample populations are generated, the next step is to analyze these populations for any change in characteristics. In this chapter, we advocate the use of statistical hypothesis tests for comparing populations. The null hypothesis in statistical tests states that distributions (or means, or std. deviations) of the two sample populations are equal. Depending on desired assumptions and the focus of analysis, different statistical tests can

be used.

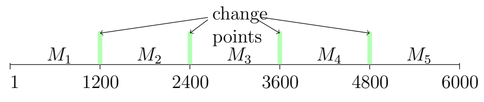
- *Interactive Visualization:* The results of comparative studies on the populations of trace characteristics can be intuitively presented to an analyst. For example, the significance probabilities of the hypothesis tests can be visualized as a drift plot. Troughs in such a drift plot signify a change in the significance probability thereby implying a change in the characteristics of traces.
- *Analyze Changes:* Visualization techniques such as the drift plot can assist in identifying the change points. Having identified that a change had taken place, this step deals with techniques that assist an analyst in characterizing and localizing the change and in discovering the change process.

5.6 Experiments and Discussion

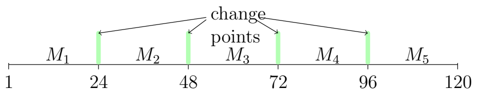
We now put the ideas proposed for handling concept drifts in practice. We illustrate change point detection and change localization in an event log containing the process variants discussed as the running example in Section 5.1. We have modeled each of these five process variants in CPN tools [180] and simulated 1200 traces for each model. We created an event log $\mathcal{L}$ consisting of 6000 traces by juxtaposing each set of the 1200 traces. The event log contains 15 activities or event classes (i.e., $|\mathcal{A}| = 15$) and $58,783$ events.

5.6.1 Change Point Detection

Given this event log $\mathcal{L}$, *our first objective is to detect the four change points pertaining to these five process variants* as depicted in Figure 5.10(a). Global features can be applied only at the log level; to facilitate this, we have split the log into 120 sub-logs using a split size of 50 traces. In this scenario, the four change points corresponding to the five process variants are as depicted in Figure 5.10(b).



(a) trace level



(b) sub-log level

Figure 5.10: Event log with traces from each of the five models juxtaposed. Also indicated are change points between models both at the trace level and at the sub-log level. The event log is split into 120 sub-logs, each containing 50 traces.

We have computed the follows relation type count (RC) of all 15 activities thereby generating a multi-variate vector of 45 features for each sub-log. We have applied the Hotelling T^2 hypothesis test on this multi-variate dataset using a moving window population of size, $w = 10$. For this hypothesis test, we have randomly chosen 12 of the 45 features with a 10-fold cross validation[4]. Figure 5.11(a) depicts the average significance probability of the Hotelling T^2 test for the 10 folds on this feature set. The troughs in the plot signify that there is a change in the distribution of the feature values in the log. In other words, they indicate that there is drift (change) in the concept, which here corresponds to the process. It is interesting to see that the troughs are observed around indices 24, 72, and 96 which are indeed the points of change (remember that we have split the log into 120 sub-logs with the change points at indices 24, 48, 72, and 96). The change at index 48 corresponding to the transition from M_2 to M_3 could not be uncovered using this feature set due to the fact that the relation type counts would be alike for logs generated from these two process variants.

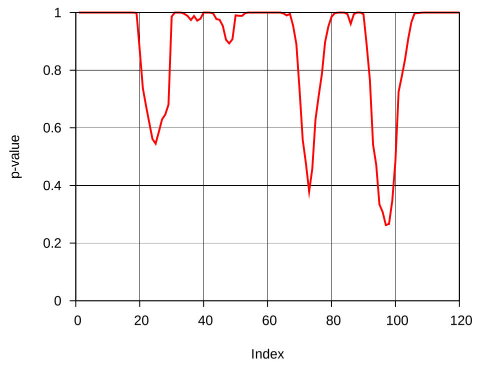
We have considered the J-measure for each sub-log and for every pair of activities, a and b in $\mathcal{A}$ (b follows a within a window of length $l = 10$). The univariate Kolmogorov-Smirnov (KS) and the Mann-Whitney U (MW) tests using a population of size $w = 10$ are applied on the J-measure of each activity pair. Figure 5.11(b) depicts the average significance probability of the KS-test on all activity pairs while Figure 5.11(c) depicts the same for the MW-test. We can see that significant troughs are formed at indices 24, 48, 72, and 96 which correspond to the actual change points. Unlike the relation type count feature, the J-measure feature is able to capture all the four changes in the models. This can be attributed to the fact that the J-measure uses the probability of occurrence of activities and their relations. In M_2, there could be cases where all the modes of notification are skipped (XOR construct). However, in M_3 at least one of the modes need to be executed (OR construct). This results in a difference in the distribution of activity probabilities and their relationship probabilities which is elegantly captured by the J-measure. As mentioned earlier, the MW-test is computationally more expensive than the KS-test as the values in each population need to be sorted (to estimate the rank-ordering).

We have considered the J-measure for each trace separately instead of at the sub-log level. Each activity pair generates a vector of dimension 6000 corresponding to the J-measure of that activity pair in each trace. The univariate Kolmogorov-Smirnov test using a population size of $w = 400$ is applied to the vector corresponding to each activity pair in $\mathcal{A} \times \mathcal{A}$. Figure 5.12(a) depicts the average significance probability of KS-test on all activity pairs while Figure 5.12(b) depicts the average significance probability of KS-test on all activity pairs using the window count feature set (WC). We can see that significant troughs are formed at indices 1200, 2400, 3600, and 4800. These are indeed the points where the models have been changed.

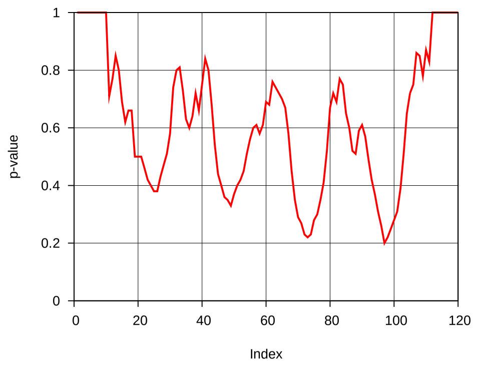
Influence of Population Size

It is imperative to see that the goodness of the results of hypothesis tests depends on the population size. The statistical analysis assumes that each population is

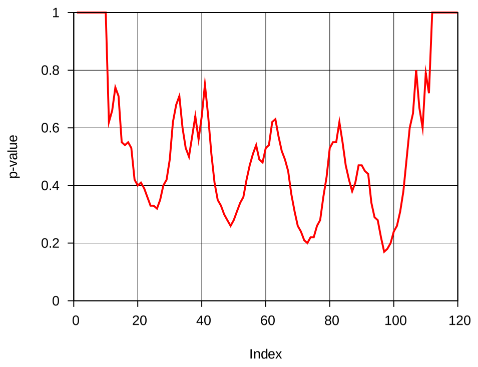
[4]The random selection of a subset of features is primarily for two reasons (i) to deal with the curse of dimensionality and (ii) the changes being centered around a few activities are prominently reflected only in those features corresponding to these activities.



(a) Hotelling T^2 Test on f_{RC}

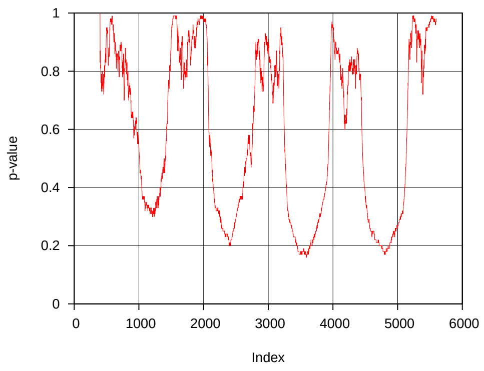


(b) KS-Test on f_J

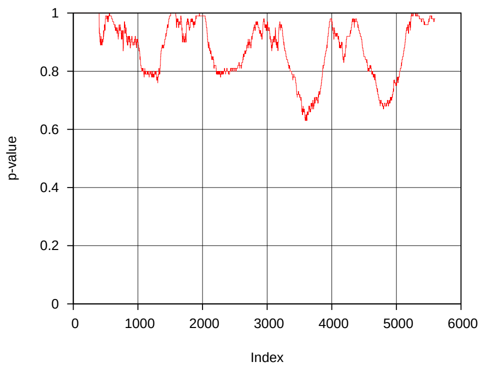


(c) MW-Test on f_J

Figure 5.11: (a) Significance probability of Hotelling T^2 test on relation counts (b) Average significance probability (over all activity pairs) of KS-test on J-measure. (c) Average significance probability (over all activity pairs) of MW-test on J-measure. The event log is split into sub-logs of 50 traces each. X-axis represents the sub-log index. Y-axis represents the significance probability of the test. Troughs signify change points.



(a) KS-Test on f_J



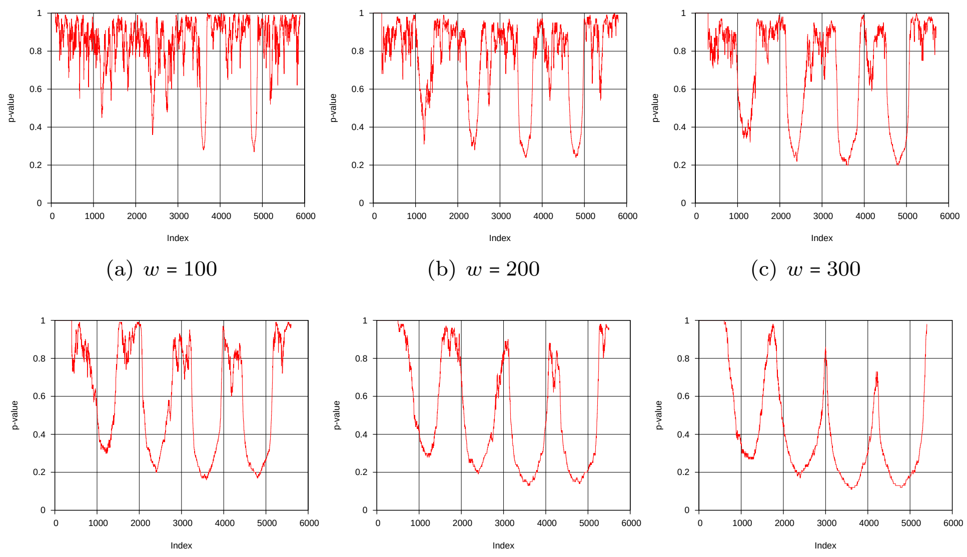
(b) KS-Test on f_{WC}

Figure 5.12: Average significance probability (over all activity pairs) of KS-test on the J-measure and WC feature sets estimated for each trace. X-axis represents the trace index. Y-axis represents the significance probability of the test. Troughs signify change points.

independent. A good population size is largely dependent on the application and the focus of analysis. In order to study the influence of population size, we have considered the J-measure for every pair of activities and the univariate KS-test for change point detection. Figure 5.13 depicts the results for varying sizes of the population. We see a lot of noise for small populations and the drift tends to be smooth as the population size increases. This can be attributed to the fact that as the population size increases (i.e., as we consider more cases), the variability in the nature of cases reduces and attains a stability, e.g., there can be a flux of low-insurance claims initially and after a certain time period the proportion stabilizes.

5.6.2 Change Localization

Our second objective in handling concept drifts is that of *change localization.* In order to localize the changes (identify the regions of change), we need to consider each activity pair individually or a subset of activity pairs. For example, the change from M_1 to M_2 is localized in the region pertaining to high insurance claim checks. We expect characteristic changes in features pertaining to these activities and other activities related to these activities. For example, in M_1, the activities `High Medical History Check` and `Contact Hospital` always follow the activity `Register` whenever a claim is classified as high. In contrast, in M_2, these activities need not always follow `Register` due to the fact that both these activities are skipped if `High Insurance Check` fails while `Contact Hospital` is skipped if `High Medical History Check` fails. During simulation, we have set the probability of success of a check to 90%. We have considered the window count (WC) feature for the activity relation `Contact Hospital` follows `Register` on a window length of $l = 10$ in each trace separately. Figure 5.14(a) depicts the significance probability of the univariate KS-test using a population size of $w = 200$ on this feature. We can see that one dominant trough is formed at index 1200 indicating that there exists



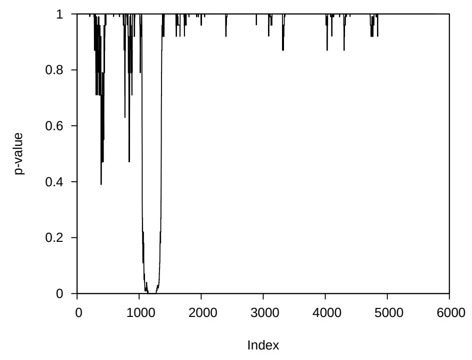
(a) $w = 100$ (b) $w = 200$ (c) $w = 300$

(d) $w = 400$ (e) $w = 500$ (f) $w = 600$

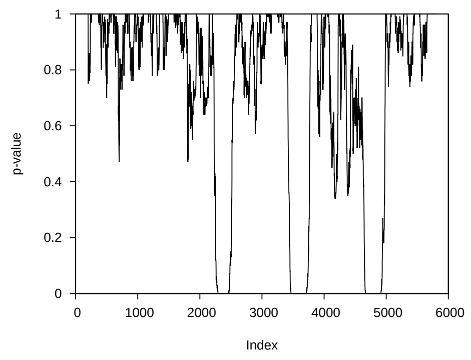
Figure 5.13: Average significance probability (over all activity pairs) for different population sizes of KS-test on the J-measure estimated over all activity pairs in each trace. X-axis represents the trace index. Y-axis represents the significance probability of the test. Troughs signify change points.

a change in the region between `Register` and `Contact Hospital`. No subsequent changes with respect to this activity pair can be noticed, which is indeed the case in the sequence of models used.

As another example, we have considered the activity `Prepare Notification` along with all the three `Send Notification` activities. There exists a change pertaining to these activities between models M_2 and M_3, M_3 and M_4, and M_4 and M_5. More specifically, we have considered the window count feature on the activity relations `Send Notification By Phone` follows `Prepare Notification`, `Send Notification By email` follows `Prepare Notification` and `Send Notification By Post` follows `Prepare Notification`. Figure 5.14(b) depicts the average significance probability of the univariate KS-tests using a population size of $w = 200$ on the WC feature of these three activity pairs. We see three dominant troughs around indices 2400, 3600, and 4800 signifying the changes in the models. Certain false alarms (minor troughs) can also be noticed in this plot. One means of alleviating this is to consider only those alarms with an average significance probability less than a threshold, δ. Another means is to consider a larger population size. Figure 5.15 depicts the average significance probability of the univariate KS-tests using a population size of $w = 400$ on the WC feature set. We can see that the noise has subsided significantly. In this fashion, by considering activities (and/or activity pairs) of interest, one can localize the regions of change. Furthermore, using this approach, one can get answers to diagnostic questions such as "*Is there a change with respect to activity* `a` *in the*

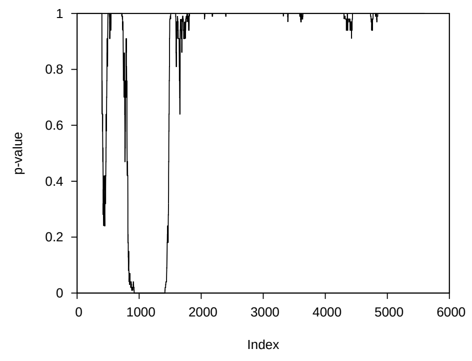


(a) `Register` and `Contact Hospital`; population size, $w = 200$

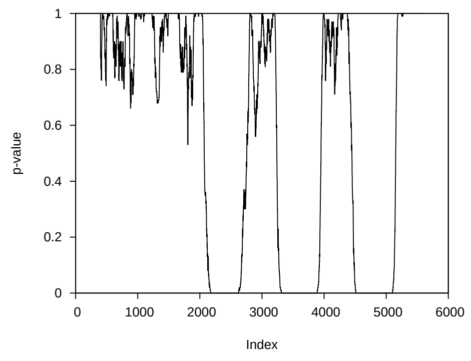


(b) `Prepare Notification` and `Send *`; population size, $w = 200$

Figure 5.14: (a) Significance probability of *KS*-test on *WC* feature estimated for the relation, `Contact Hospital` follows `Register`. Trough indicates change point w.r.t this feature. (b) Average significance probability (over activity pairs) of *KS*-test on *WC* feature estimated for the various modes of `Send Notification` follows `Prepare Notification` relation. Troughs indicate change point w.r.t these activities. X-axis represents the trace index. Y-axis represents the significance probability of the test.



(a) `Register` and `Contact Hospital`; population size, $w = 400$

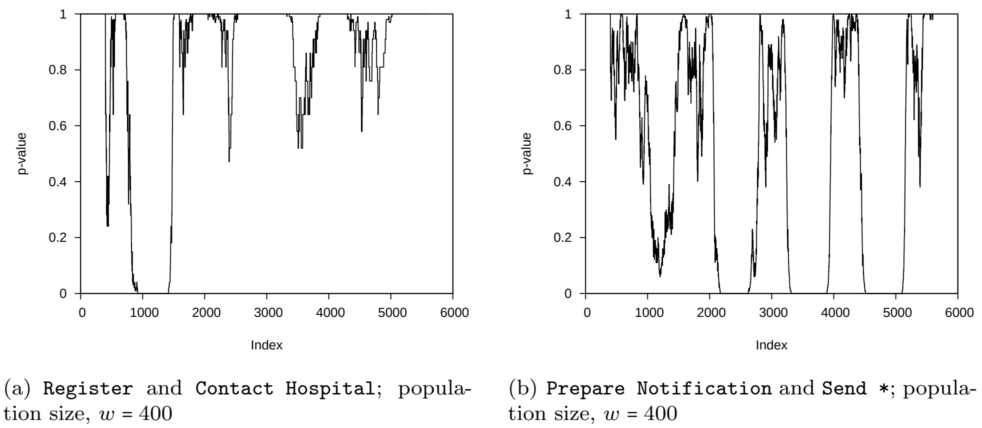


(b) `Prepare Notification` and `Send *`; population size, $w = 400$

Figure 5.15: (a) Significance probability of *KS*-test on *WC* feature estimated for the relation, `Contact Hospital` follows `Register`. Trough indicates change point w.r.t this feature. (b) Average significance probability (over activity pairs) of *KS*-test on *WC* feature estimated for the various modes of `Send Notification` follows `Prepare Notification` relation. Troughs indicate change point w.r.t these activities. X-axis represents the trace index. Y-axis represents the significance probability of the test.

process at time period t?".

The window count feature (WC) performs better in change localization in comparison to the J-measure. This is due to the fact that the J-measure uses the probability of activities which can be affected due to changes anywhere in the pro-



(a) `Register` and `Contact Hospital`; population size, $w = 400$

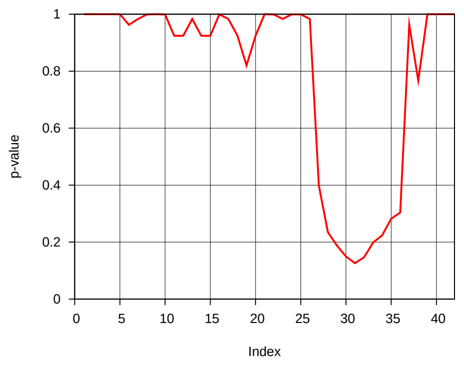
(b) `Prepare Notification` and `Send *`; population size, $w = 400$

Figure 5.16: (a) Significance probability of KS-test on the J-measure estimated for the relation, `Contact Hospital` follows `Register`. (b) Average significance probability (over activity pairs) of KS-test on the J-measure estimated for the various modes of `Send Notification` follows `Prepare Notification` relation.

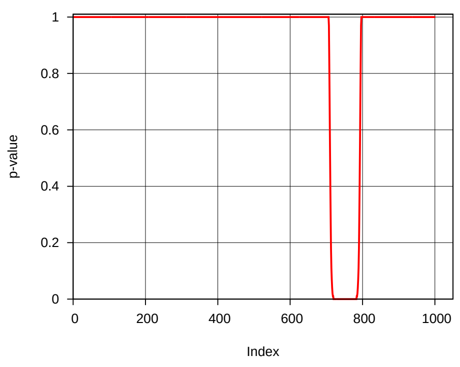
cess irrespective of our region of focus. For example, consider the J-measure for the relation `Contact Hospital` follows `Register`. The probability of occurrence of both `Register` and `Contact Hospital` is affected by the changes in the process model corresponding to the sending of notifications as well, e.g., in M_3 since all the modes of send notification are executed, the probability of `Contact Hospital` in a trace is smaller than a corresponding trace (`Contact Hospital` is executed) in M_4 where only one of the notifications is possible. Figure 5.16(a) depicts the significance probability of the univariate KS-test on the J-measure for the activity relation `Contact Hospital` follows `Register` while Figure 5.16(b) depicts the average significance probability of the univariate KS-tests on the J-measure of various `Send Notification` modes following `Prepare Notification` using a population size of $w = 400$. As we can see, using the J-measure leads to identifying any change happened in the process and not clearly in localization. Therefore, *we recommend the use of window count feature for change localization.*

Digital Copier Example

As another example, we report the results of applying the concepts proposed in this chapter on the simple digital copier. For this purpose, let us assume that the first version of the digital copier did not have the support for the zooming functionality (in `Image Processing`); the product was enhanced with this functionality in its second version. We consider the copy/scan subprocess of the copier for these two versions. Note that this process has a lot more variability than the insurance claim process, e.g., loop constructs with the number of iterations depending on the number of pages to copy/scan (in `Capture Image`) or until the error reaches a threshold (in `Half Toning`), parallelism constructs, etc. We have modeled the two variants using CPN tools [180] and simulated 750 and 300 traces respectively. The traces pertain



(a) change point detection using sub-logs



(b) change localization w.r.t the activity pair `Photo Quality Reproduction` and `Zooming Complete`

Figure 5.17: (a) Significance probability of the Hotelling T^2 test on the RC feature on the sub-logs using a split size of 25 traces (b) Significance probability of the KS-test on WC feature estimated for the relation, `Zooming Complete` follows `Photo Quality Reproduction` in each trace.

to copy/scan requests with up to five pages. We create an event log containing 1050 traces by juxtaposing the traces corresponding to the two variants. This event log contains 47, 726 events distributed over 35 activities. We have considered the follows relation type count feature over the sub-logs with a split size of 25 traces. This feature type generates 105 features (3×35) for each sub-log. Therefore, the data set contains a multi-variate vector of 42 elements (for the 42 sub-logs) with each element containing 105 features. We randomly considered five features with a 20-fold cross validation. Figure 5.17(a) depicts the average significance probability of the Hotelling T^2 test for the 20 folds on this feature set using a population size of $w = 5$. As we can see, a drift has been clearly detected around index 30 corresponding to the point where the first 750 traces have been executed. In order to localize the change, we considered the WC feature for the activity pair `Photo Quality Reproduction` and `Zooming Complete` (in the `Image Processing` subprocess) in each trace. The data set contains 1050 elements with each element being a scalar value (corresponding to the window count of the chosen activity pair). Figure 5.17(b) depicts the result of applying the KS-test on this data set using a population of size 50. We can see that a drift has been detected with respect to this activity pair, which is indeed the case (recall that the first version of the copier does not support zooming functionality).

Our experiments substantiate that the proposed feature sets and the framework for handling concept drifts has significant promise in detecting changes and change points, and in identifying the regions of change.

5.7 Outlook

Dealing with concept drifts raises a number of scientific and practical challenges. In this section, we highlight some of these challenges.

- *Change-pattern Specific Features:* In this chapter, we presented very generic features (based on follows/precedes relation). These features are neither complete nor sufficient to detect all classes of changes. An important direction of research would be to define features catering to different classes of changes and investigate their effectiveness. A taxonomy/classification of change patterns and the appropriate features for detecting changes with respect to those patterns is needed. For example, if we would like to detect changes pertaining to a loop construct (insertion/removal/modification of loops as changes in process variants), tandem arrays (cf. Chapter 3) would be an appropriate feature to consider.
- *Feature Selection:* The feature sets presented in this chapter result in a large number of features. For example, the activity relation count feature type generates $3 \times |\mathcal{A}|$ features whereas the window count and J-measure generate $|\mathcal{A}|^2$ features (corresponding to all activity pairs). On the one hand, such high dimensionality makes analysis intractable for most real-life logs. On the other hand, changes being typically concentrated in a small region of a process make it unnecessary to consider all features. There is a need for dimensionality reduction techniques [71, 99] that can efficiently select the most appropriate features.
- *Holistic Approaches:* In this chapter, we discussed ideas on change detection and localization in the context of sudden changes to the control-flow perspective of a process. However, as mentioned in Section 5.3, the data and resource perspectives are also equally important. So are the contexts of gradual drifts. Features and techniques that can enable the detection of changes in these other perspectives need to be discovered. Furthermore, there could be instances where more than one perspective (e.g., both control and resource) change simultaneously. Hybrid approaches considering all aspects of change holistically need to be developed.
- *Techniques for Drift Detection:* In this chapter, we explored just the Hotelling's T^2 test to deal with multi-variate data. In addition, we have dealt with multiple features by considering univariate hypothesis tests on each feature separately and averaging the test results over all features. However, further investigation needs to be done on hypothesis tests devised naturally for multi-variate data. Also, alternatives to hypothesis testing that can uncover drifts and diagnose the changes are a welcome addition to the repertoire of techniques for handing concept drifts in process mining.
- *Unraveling Process Evolution:* As mentioned earlier, after detecting the change points and the regions of change, it is necessary to put them together in perspective. Organizations would be interested in discovering the evolution of change (for example, as an animation depicting how the process has changed/evolved over time). In addition, there are other applications such as deriving a configurable model for the process variants. A *configurable process model* describes

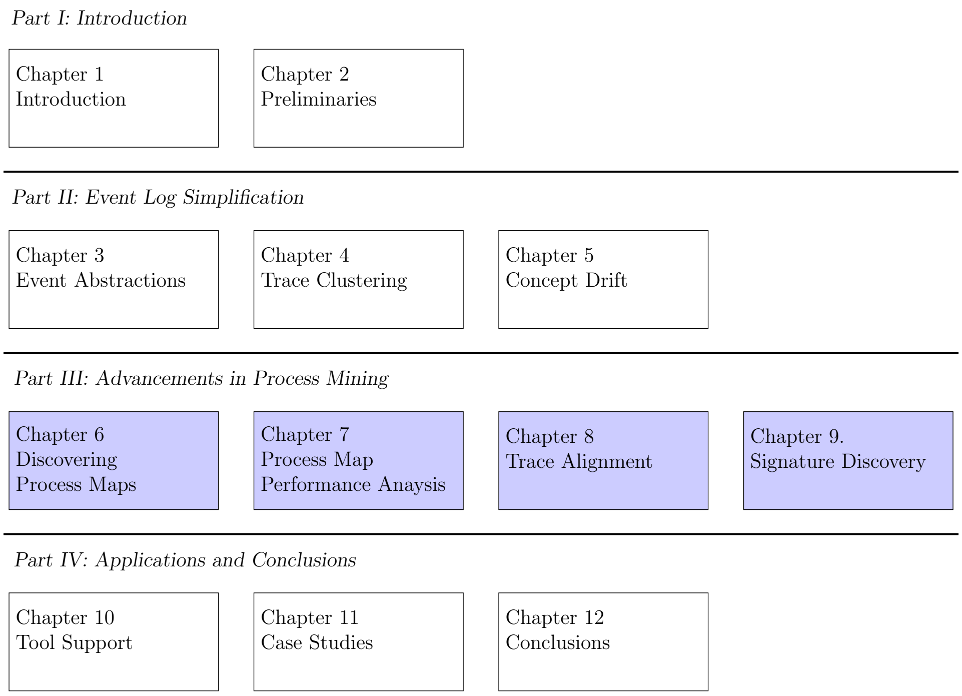
a family of similar process models [233]. The process variants discovered using concept drift can be merged to derive a configurable process model.

- *Sample Complexity:* Sample complexity refers to the number of traces (size of the event log) needed to detect, localize, and characterize changes within acceptable error bounds. This should be sensitive to the variability in processes (in the manifestation of various process model constructs used), nature of changes, their influence and manifestation in traces, and the feature space and algorithms used for detecting drifts. On a broader note, the topic of sample complexity is relevant to all facets of process mining and is hardly addressed. For example, it would be interesting to know the lower bound on the number of traces required to discover a process model with a desired fitness.
- *Online (On-the-fly) Drift Detection:* In this chapter, we have looked at detecting drifts in an offline setting, i.e., for postmortem analysis. Although detecting concept drifts is important for off-line analysis, it is more interesting and appropriate for online analysis, e.g., an organization would be more interested in knowing a change in behavior of their customers or a change in demand *as and when* it is happening. Such near real-time triggers (alarms) will enable organizations to take quick remedial actions and avoid any repercussions. We believe the proposed framework to be applicable even for online analysis. However, few new challenges emerge, e.g., the number of samples required remains an issue. In addition, one needs additional computational power and efficient techniques to do such analysis in real-time.

5.8 Conclusions

This chapter introduced the topic of concept drift in process mining, i.e., analyzing process changes based on event logs. We proposed feature sets and techniques to effectively detect the changes in event logs and identify the regions of change in a process. Our initial results show that heterogeneity of cases arising due to process changes can be effectively dealt with by detecting concept drifts. Once change points are identified, the event log can be partitioned and analyzed. This is the first step in the direction of dealing with changes in any process monitoring and analysis efforts. We have considered changes only with respect to the control-flow perspective manifested as sudden drifts. Therefore, our analysis should only be seen as the starting point for a new subfield in the process mining domain.

Part III
Advancements in Process Mining


Part I: Introduction
Chapter 1 Introduction
Chapter 2 Preliminaries
Part II: Event Log Simplification
Chapter 3 Event Abstractions
Chapter 4 Trace Clustering
Chapter 5 Concept Drift
Part III: Advancements in Process Mining
Chapter 6 Discovering Process Maps
Chapter 7 Process Map Performance Anaysis
Chapter 8 Trace Alignment
Chapter 9. Signature Discovery
Part IV: Applications and Conclusions
Chapter 10 Tool Support
Chapter 11 Case Studies
Chapter 12 Conclusions

In the previous part, we focused on preprocessing techniques leading to simplification of event logs. Though log simplification is important and necessary, it alone may not be sufficient. In this part, we focus on the second aspect, *advancements in process mining*, which addresses some of the challenges in process mining, viz., dealing with less-structured processes and provisions for process diagnostics. Chapter 6 presents a two-phase approach that enables the discovery of hierarchical process models. Chapter 7 presents techniques that assist in identifying bottlenecks in processes by replaying an event log on a process model and computing performance measures. Chapter 8 presents a technique of aligning traces so that event logs can be explored easily. Trace alignment assists in answering a variety of diagnostic questions and extends the scope of conformance checking to also cover the direct inspection of traces. Chapter 9 presents a diagnostic technique of finding signature patterns that can discriminate between different classes of behavior in event logs.

Chapter 6
Discovering Process Maps

In the previous chapters, we have seen techniques for dealing with fine granular events, heterogeneity in event logs, and process changes. Such characteristic features are highly relevant for large scale event logs. Our techniques help in simplifying event logs through preprocessing and, thus, assist in improving process mining results. Though log simplification is a critical step, it alone is not sufficient to provide direct and comprehensible answers to analysts exploring process mining, e.g., process models mined from simplified event logs can still be spaghetti-like. The inability to provide answers is due to the unavailability of a suitable technique, e.g., dealing with less structured processes, provisions for process diagnostics such as uncovering patterns that can discriminate between different classes of behavior, for instance, patterns that can discriminate between good insurance claims and fraudulent insurance claims, etc. In the coming chapters, we focus on extending process mining research to deal with some of the shortcomings (e.g., dealing with less-structured and spaghetti-like processes, provisions for process diagnostics, etc.) and explore new techniques in analyzing event logs. In this chapter, we focus on one of the three pillars of process mining, viz., process discovery.

Process discovery is one of the three main types of process mining [221]. A discovery technique takes an event log and produces a model without using any *apriori* information, e.g., the α-algorithm discovers a Petri net based on sequences of events [229]. Practical experiences of applying traditional process discovery techniques revealed two problems: (a) processes tend to be less structured than what stakeholders expect and (b) events logs contain fine-grained events whereas stakeholders would like to view processes at a more coarse-grained level. These problems are more pronounced in event logs from high-tech systems. More often than not the mined models are complex and unsatisfactory (spaghetti-like). This can be attributed to the fact that the majority of process discovery techniques in the literature pertain to the discovery of control-flow models that are "flat" [229, 238, 246, 264]. A notable exception is the Fuzzy miner [94]. Flat (and complex) models have a negative influence on model comprehension and analysis. For a log with $|\mathcal{A}|$ event classes (activities), a flat model can be viewed as a graph containing $|\mathcal{A}|$ nodes with edges corresponding to the causality defined by the execution behavior in the log. Graphs become quickly overwhelming and unsuitable for human perception and cognitive systems even if there are a few dozens of nodes [83].

Process models can be seen as “maps” describing the operational processes of organizations. Analogous to cartography, process discovery techniques should allow for various context-dependent views on these process maps. For example, the perspective of analysis may be different depending on someone’s role and expertise; a manager may be interested in a high level view while a specialist may be interested in a detailed analysis of some process fragment. Process discovery techniques should facilitate the extraction of process maps eliciting the respective desired traits and hiding the irrelevant ones for various users. Furthermore, these techniques should uncover comprehensible models by providing hierarchical views with a facility to seamlessly zoom-in and/or zoom-out.

In Chapter 3, we showed that common execution patterns (e.g., tandem arrays, maximal repeats, etc.) manifested in an event log can be used to create powerful abstractions. In this chapter, we use these abstractions and propose a *two-phase approach to process discovery*. The first phase comprises of pre-processing the event log based on abstractions (bringing the log to the desired level of granularity), and the second phase deals with discovering the process maps while providing a seamless zoom-in/out facility. We demonstrate that we can use this approach to create maps that (i) *depict desired traits*, (ii) *eliminate irrelevant details*, (iii) *reduce complexity*, and (iv) *improve comprehensibility*. Figure 6.1 summarizes the overall approach.

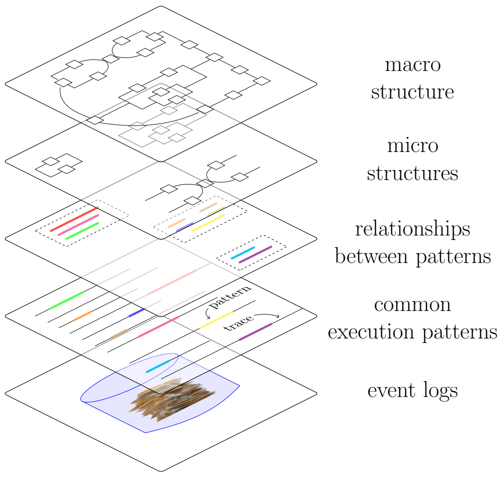


Figure 6.1: Repeating subsequences of activities define the common execution patterns and carry some domain (functional) significance. Related patterns and activities pertaining to these patterns define abstractions that correspond to micro-structures (or subprocesses). The top-level process model can be viewed as a macro-structure that subsumes the micro-structures.

The remainder of this chapter is organized as follows. Related work on mining process models from event logs is presented in Section 6.1. Section 6.2 discusses the two-phase approach to process discovery. Section 6.3 utilizes the abstractions defined over common execution patterns (as discussed in Chapter 3) and presents an algorithm to transform an event log to a desired level of granularity. Section 6.4 describes how the transformation of logs can be used to derive a process model at a particular level. We present and discuss some experimental results in Section 6.5. The scalability of the proposed approach is discussed in Section 6.6. Section 6.7 presents the limitations and extensions of the proposed approach. Finally, Section 6.8 concludes the chapter.

6.1 Overview of Approaches to Process Discovery

Process discovery is concerned with discovering a process model from example behavior recorded in an event log. In this chapter, we focus on the control-flow perspective and propose a two-phase approach to process discovery that enables the mining of hierarchical process models. Dozens of techniques have been proposed for control-flow mining since the late nineties. We review these techniques in this section. The techniques primarily differ in two aspects: (i) the formalism used to represent processes and (ii) the specific algorithms used to discover them. Process models can be represented in several ways, ranging from directed graphs to more expressive formalisms such as the ones presented in Section 2.7.

Process discovery from event logs has been investigated by Cook et al. [38, 39, 41] in the context of software engineering. They extended and developed algorithms to mine sequential patterns from event logs (capturing the software development processes) and proposed three methods, viz., *RNet*, *KTail*, and *Markov*, for representing them as Finite State Machines (FSMs). Their objective was to mine a model that expresses the most frequent patterns in the log rather than a model that is complete and precise. The *RNet* algorithm, based on neural nets, considers a window of predecessor events and statistically determines the succession of events in the FSM. In contrast, the *KTail* algorithm analyzes a window of successor events to create the process model. The Markov algorithm takes the advantages of both worlds, i.e., considers both the predecessors and successors of events, for determining the order of tasks in the process model. In [40, 42], Cook et al. extend their Markovian approach to mine concurrent process models. They propose four metrics to identify the nature of splits/join (XOR and AND constructs are tackled). These algorithms are robust to noise due to their probabilistic nature. The main drawback of their work is that it does not yield explicit process models, but rather a set of dependency relations between event types.

Process discovery in the context of business process management was first explored by Agrawal et al. [6]. They call it *workflow mining* and discover workflow graphs that indicate the dependencies between tasks in the process. The algorithm assumes the log to contain atomic events (single event type) and that the process model has an unique start task and an unique end task. Furthermore, the algorithm does not handle duplicate tasks and assumes that no task appears more than once in a process instance. However, loops are tackled by a combination of re-labeling the log prior to mining and folding explicitly detected iterations in the end. There is no indication of the semantics of the split/join points. However, there exists edge (arc) and join conditions, which can be discovered through data mining classification algorithms.

Two types of tokens, viz., true and false, are propagated through the graph. Only true tokens can trigger the execution of a task, while false tokens are simply passed through. The types of tokens produced by a task execution are specified by arc and join conditions. The mined workflow graph is able to generate the sequences of events found in the event log. Pinter et al. [81, 173] extended the work of Agrawal et al. [6]. They allow event logs with two event types, indicating the start or completion of a task, and use this information to derive explicit parallel relations between activities. They analyze the interleaving of time spans in which activities have been executed to infer concurrency/parallelism.

Herbst et al. [106–108] propose three two-step inductive algorithms, viz., *MergeSeq*, *SplitSeq*, and *SplitPar* for process discovery. In the first step, a *Stochastic Activity Graph* (SAG) is mined that captures the dependencies between the tasks in the workflow log. Transition probabilities indicating the probability of one task following the other are also estimated. The second step converts the SAG to the Adonis Definition Language (ADL), which is a block-structured language to specify workflow models. The conversion from SAG to ADL aims at creating well-defined workflow models. The mining of sequential models are dealt with MergeSeq and SplitSeq [107] while concurrent models are dealt with SplitPar [106, 108]. A noteworthy aspect of these algorithms is that they can deal with event logs containing duplicate tasks, i.e. there may be multiple task nodes in the process model with the same label.

Van der Aalst et al. [225, 229, 242] have proposed the α-algorithm to discover Petri nets from event logs. The α-algorithm is based on four binary relations derived locally between tasks in the log, viz., *follows*, *causal*, *parallel*, and *unrelated*. Two tasks `a` and `b` are said to have a *follows* relation if they appear next to each other in the log. They are said to be *unrelated*, otherwise. The tasks `a` and `b` are said to have a

- *causal* relation if `a` directly follows `b`, but `b` does not directly follow `a`
- *parallel* relation if `a` directly follows `b` and also `b` directly follows `a`

Under the assumption that the event logs are complete (with respect to the *follows* relation) and noise-free, the α-algorithm has been proven to work for the subclass of Structured Workflow Nets (SWF nets) without short loops and implicit places [229]. The α-algorithm does not consider the frequency of a relation but only checks if the relation holds or not. This makes the approach sensitive to noise. Furthermore, the α-algorithm cannot tackle duplicate tasks. The α-algorithm has been extended in [51, 52, 265] to tackle short loops and in [266] to be able to mine Petri nets with local or non-local non-free choice constructs. Weijters et al. [264] propose an algorithm, called *Heuristics Miner*, that uses the follows relation of the α-algorithm but infers the other three relations (causal, parallel, and unrelated) by considering the frequency of the follows relation in the log. The mined models are called as *heuristic nets*. Since this approach considers the frequency of relations, it can handle noise.

Alves de Medeiros et al. [49, 53, 231] propose a process discovery approach based on genetic algorithms. The basic idea of this approach is to start from an initial population of individuals (process models) that are candidates for the solution. Heuristic nets are used to represent the population of process models. From the initial population, the genetic algorithm iteratively converges to an appropriate solution, mimicking the process of evolution in nature, by using the genetic operations, *cross-over* and *mutation*. In each iteration, the algorithm generates candidate population (children) from the current population by using the cross-over and mutation operations, and selects the best candidates for the next iteration. The selection mechanism is guided by various quality criteria such as the *fitness* of the models on the input event log. The algorithm terminates with the best individual (process model with the highest fitness) as the result upon reaching a *stop criterion*. This approach can deal with long-term dependencies and duplicate tasks and is robust to noise.

Van Dongen et al. [243, 245] propose a multi-phase approach to mine Event Driven Process Chains (EPCs) from event logs. The first phase consists of mining a process model (EPC) for every trace in the event log. These models show the partial order between the instances of tasks in a trace. The second phase involves the aggregation (or merging) of the mined models, for every trace, during the first phase. This approach distinguishes between three types of split/join points, viz., AND, OR, and XOR, based on frequencies associated to the edges of the aggregated task using the following straight forward rules:

- if a split point occurs as often as its direct successors, it is assumed to be of type AND,
- if a split point occurs as often as the sum of occurrences of its direct successors, it is assumed to be of type XOR,
- otherwise, the split is assumed to be of type OR.

Process discovery based on the *theory of regions* [11] has been proposed in [13, 31, 125, 206, 232, 238, 239]. The theory of regions deals with the construction (synthesis) of Petri nets from a description of its observed behavior. There are two types of regions: (i) *state-based regions* and (ii) *language-based regions*. State-based region theory uses transition systems to express the observed behavior while language-based region theory expresses the observed behavior in the form of a prefix-closed language (e.g., regular language). Kindler et al. [125] first create a transition system from an event log and then use the state-based region theory to mine Petri nets. Van der Aalst et al. [232] extend this approach by introducing a methodology for deriving an input transition system. Furthermore, this approach supports choosing the level of abstraction from the observed behavior, i.e., how much the mined Petri net generalizes from the observed behavior. Bergenthum et al. [13] and Van der Werf et al. [238] use the language-based region theory to mine Petri nets. These approaches though efficient when compared to state-based region approaches (since we avoid the construction of a transition system) suffer from the lack of configurability, e.g., in terms of the desired level of abstraction.

All of the above approaches to process discovery mine models that are “flat”. Flat models become hard to comprehend, especially in case of less structured processes, and large scale and heterogenous event logs. Greco et al. [86, 87, 89] proposed an approach to mine hierarchies of process models that collectively represent the process at different levels of granularity and abstraction. The basic idea of their approach is to cluster the event log into different partitions based on the homogeneity of traces and mine process models for each of the clusters. Clustering induces a hierarchy in the form of a tree with the root node depicting the entire log and the leaf nodes corresponding to traces pertaining to concrete usage scenarios. Two kinds of abstractions, viz., *is-a* and *part-of*, is defined over activities. The abstraction type is determined by traversing the tree bottom-up and in the process checking whether a pair of activities can be merged without adding too many spurious control-flow paths among the remaining activities. This approach tries to analyze the mined process models (post-processing) for identifying activities that can be abstracted. In contrast, the two-phase approach to process discovery that we propose in this chapter involves analyzing the raw traces and defining abstractions (pre-processing).

Another class of approaches for the abstraction of process models deals with the preservation of the main (significant) elements while discarding insignificant ones. Taking cartography as a metaphor, Günther and Aalst [94] have proposed a process mining approach to deal with the "spaghettiness" of less structured processes. The basic idea here is to assign significance and correlation values to activities and transitions, and depict only those edges/activities whose significance/correlation is above a certain threshold. Less significant activities/edges are either removed or clustered together in the model. On similar lines, Polyvyanyy et al. [175] have proposed a slider approach for enabling flexible control over various process model abstraction criteria (such as activity effort, mean occurrence of an activity, and probability of a transition). The slider is employed for distinguishing significant process model elements from insignificant ones. Approaches such as [94, 175] look at abstraction from the point of retaining highly significant information and discarding less significant ones in the process model. In contrast, the approach that we propose looks at abstraction from a functionality point of view.

The first step of the two-phase approach to process discovery that we propose in this chapter is primarily a preprocessing step on the event logs to take them to a desired level of granularity. The second phase involves the mining of process models over the preprocessed event logs. Any discovery approach can be used to mine the models at a given level.

6.2 Two-phase Approach to Process Discovery

In this section, we present our two-phase approach to process discovery. The first phase involves the simplification of the log to a desired level of granularity and the second phase involves the mining of process maps.

Phase-1: Preprocessing Log

In this phase, the log is simplified based on the desired traits of the context of analysis. *Some notion of abstraction of the low-level events to a desired level of granularity needs to be defined.* In some applications and contexts, this can be provided by analysts based on domain knowledge, e.g., there can be certain activities that demarcate the execution of a particular medical procedure on an X-ray machine. The sequences of events in-between these activities pertain to an execution sequence of this procedure, e.g., the sequence of events between `Start Fluoroscopy` and `Stop Fluoroscopy` defines an instance of a `Fluoroscopy` procedure applied to a patient on an X-ray machine. Alternatively, abstractions can be defined by uncovering common execution patterns in the log as discussed in Chapter 3. These common execution patterns typically capture a subprocess/functionality. Such subprocess behavior in its totality can be captured as an *abstract activity.* Henceforth, we refer to activity sequences defining abstractions as *patterns.*

There can exist relationships between patterns that define abstractions. For example, two or more patterns can define the same abstraction, there can be permuted patterns, certain patterns can be subsequences of other patterns, etc. As discussed in Chapter 3, we can establish relationships between patterns by considering their alphabets (i.e,. the set of activities defining a pattern) and creating a Hasse diagram, i.e., a pattern graph, using the subsumption property over pattern alphabets. Figure 6.2 depicts an example of a pattern graph. The nodes in the graph correspond to pattern alphabets. Each pattern alphabet has a set of patterns associated to it, e.g., pattern alphabet {`b`, `d`, `e`, `l`} contains the sequences {`lebd`, `elbd`}.

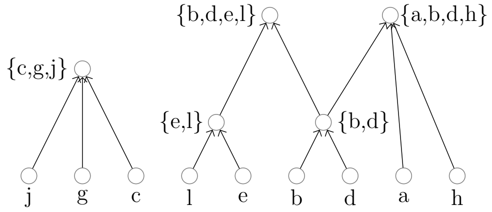
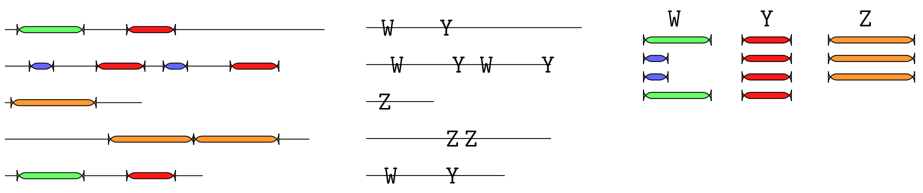


Figure 6.2: An example pattern graph establishing relationships between patterns.

Abstractions can be considered as a mapping, $\mathcal{M} \subseteq 2^{\Sigma} \times 2^{\Sigma^+} \times 2^{\mathcal{A}}$, between the *original alphabet*, Σ, of the event log, sets of patterns defining abstractions, and an *abstract alphabet* $\mathcal{A}$. An example mapping is $\mathcal{M} = \{(\{\texttt{c},\texttt{g},\texttt{j}\},\{\texttt{jgc}\},\{\texttt{W}\}),(\{\texttt{e},\texttt{l}\},\{\texttt{le}\},\{\texttt{Y}\}),(\{\texttt{b},\texttt{d},\texttt{e},\texttt{l}\},\{\texttt{lebd},\texttt{elbd}\},\{\texttt{Y}\})\}$. This mapping is analogous to the grouping and tagging of streets as a town/city in cartography and to the selection of a desired perspective of viewing maps (restaurant maps vs. fuel station maps)[1]. Each $(A,S,X) \in \mathcal{M}$ reflects a set of patterns $S \subseteq A^+$ defined by the alphabet A for the abstraction X.

[1]One can also take the analogy of *atoms* and *molecules* from chemistry. The individual activities in the original event log are atoms while the mapping associates groups of atoms (activities) to molecules (abstract activities).

Using this mapping, the original event log $\mathcal{L}$, is transformed into an *abstract log* $\mathcal{L}'$ in this phase. Each trace $\mathbf{t} \in \mathcal{L}$ is transformed into a corresponding trace $\mathbf{t}' \in \mathcal{L}'$. In each trace $\mathbf{t}$, the manifestation of each pattern $\mathbf{s} \in S$ captured by $(A, S, X) \in \mathcal{M}$ is replaced with an *abstract activity*, $\mathtt{x} \in X$, in the transformed trace. At the same time, we create one sub-log for each abstract activity $\mathtt{x}$ whose process instances correspond to the replaced pattern manifestations. Figure 6.3 depicts the general idea on the event log transformation. $\mathcal{D} = \bigcup_{(A,S,X) \in \mathcal{M}} A$ denotes the set of activities in Σ for which a mapping is defined. The activities in $\Sigma \setminus \mathcal{D}$ being not involved in the definition of mapping indicate activities that are insignificant from the context of analysis and are filtered from $\mathbf{t}$ during this transformation. We define the transformation process in detail in Section 6.3. The transformation of logs can be associated to the concept of *artifacts* [161] or *proclets* [226] in business processes. Artifact-centric process models partition a monolithic process model into smaller loosely-coupled process fragments, each describing the life-cycle of a concrete and identifiable artifact.



(a) original log with common execution patterns (b) transformed log and the sub-logs for each abstraction

Figure 6.3: Transformation of the original log into an abstracted log. Also, one sub-log is created for each abstract activity. W, Y, and Z are abstract activities.

Phase-2: Mining Maps

The second phase is to mine a process model for the abstracted (transformed) log. The mapping defined in Phase-1 induces a hierarchy over the abstract activities. Upon zooming into an abstract activity, a process model depicting the subprocess captured by this abstract activity can be shown. The sub-log for the abstract activity is used to create this subprocess model. Multiple levels of hierarchy can be obtained by a repetitive application of Phase-1, i.e., abstractions are defined over the transformed log (pre-processed log with abstractions) obtained in iteration i in iteration $i+1$. This results in new abstractions to be defined over existing abstractions, thus inducing a hierarchy. Figure 6.4 depicts this idea. The first iteration induces three abstract activities W, Y, and Z defined over common execution patterns on the input event log. The event log is transformed with these abstractions and sub-logs are created for each of the abstract activities. In the second iteration, the transformed log of iteration 1 serves as the input log. In the second iteration, an abstraction Q is defined over a pattern involving the abstract activities W and Y defined in the first iteration. In other words Q subsumes W and Y thereby inducing a second-level of hierarchy. In this fashion, one can define multiple levels of hierarchy by a repeated application of Phase-1.

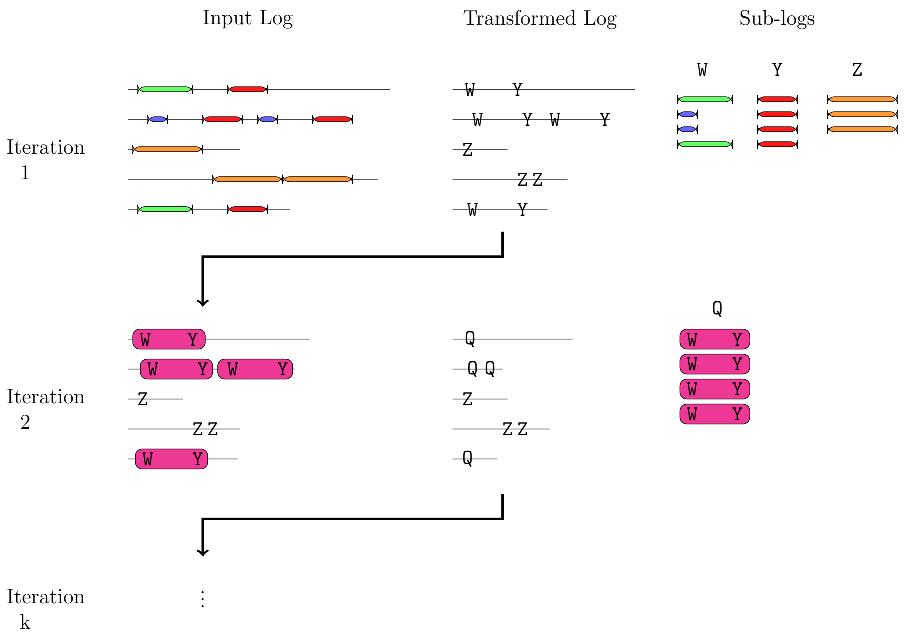


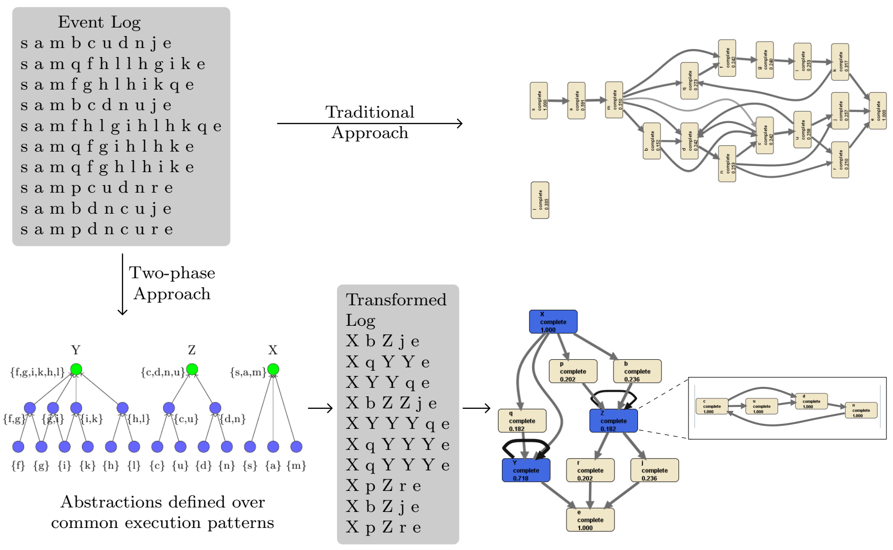
Figure 6.4: Iterative transformation of an event log induces a hierarchy of abstractions. The abstract activity `Q` introduced in iteration 2 is defined over abstract activities `W` and `Y`.

Figure 6.5 highlights the difference between the traditional approach to process discovery and our two-phase approach. Note that the process model (map) mined using the two-phase approach is simpler and that this approach enables the abstraction of activities based on functionality and provides a seamless zooming into the subprocesses captured in the abstractions.

6.3 Transforming a Log with Abstractions

In this section, we present and discuss the transformation of an event log with abstractions. We use the abstractions defined over the common execution patterns as discussed in Chapter 3. However, the proposed transformation approach can also be used for abstractions defined based on domain knowledge. Recall from Chapter 3 that abstractions are defined over the nodes in the pattern graph. This needs to be captured in the mapping $\mathcal{M} \subseteq 2^{\Sigma} \times 2^{\Sigma^+} \times 2^{\mathcal{A}}$ defined in Section 6.2. Figure 6.6 depicts an example pattern graph and an example mapping $\mathcal{M}$ based on this pattern graph. The original alphabet $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l\}$ and the abstract alphabet $\mathcal{A} = \{W, Y, Z, f, i\}$.

Algorithm 6.1 presents the approach of transforming an event log with abstractions. The basic idea of the algorithm is to scan each trace from left to right and


Traditional Approach
Two-phase Approach
Y
Z
X
{f,g,i,k,h,l}
{c,d,n,u}
{s,a,m}
{f,g}
{g,i}
{i,k}
{h,l}
{c,u}
{d,n}
{f} {g} {i} {k} {h} {l} {c} {u} {d} {n} {s} {a} {m}
Abstractions defined over common execution patterns

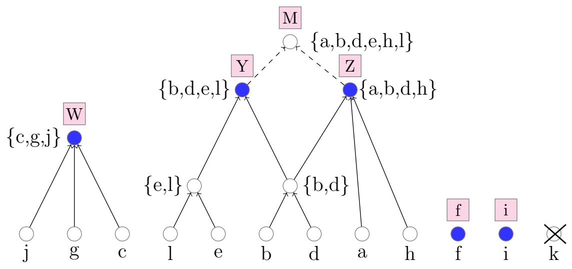
Event Log

```
s a m b c u d n j e
s a m q f h l l h g i k e
s a m f g h l h i k q e
s a m b c d n u j e
s a m f h l g i h l h k q e
s a m q f g i h l h k e
s a m q f g h l h i k e
s a m p c u d n r e
s a m b d n c u j e
s a m p d n c u r e
```

Transformed Log

```
X b Z j e
X q Y Y e
X Y Y q e
X b Z Z j e
X Y Y Y q e
X q Y Y Y e
X q Y Y Y e
X p Z r e
X b Z j e
X p Z r e
```

Figure 6.5: Traditional approach versus our two-phase approach to process discovery.



Pattern Alphabet	Patterns	Abstraction
{c, g, j}	{jgc}	W
{j}	{j}	W
{g}	{g}	W
{c}	{c}	W
{b, d, e, l}	{lebd, elbd}	Y
{e, l}	{le}	Y
{a, b, d, h}	{ahbd}	Z
{b, d}	{bd}	Y, Z
{f}	{f}	f
{i}	{i}	i

Figure 6.6: An example of a pattern graph and the mapping of pattern alphabets with abstractions.

in the process determine whether there exists a pattern in the trace for which an abstraction is defined. If such a pattern exists, the manifestation of the pattern in the trace is replaced by its abstract activity and simultaneously the manifestation of the pattern that is replaced is added as a trace in the sub-log corresponding to that abstract activity. It could be the case that the manifestation of patterns is disturbed by the presence of concurrent activities in a trace. The algorithm also considers non-continuous manifestations to deal with such scenarios. Patterns that are affected by noise, e.g., activities that are skipped, are dealt with by considering approximate manifestations. We formally define the continuous, non-continuous, and approximate manifestation of patterns as follows.

Definition 6.1 (Subsequence). A sequence $\mathbf{t}_1 = \mathbf{t}_1(1)\mathbf{t}_1(2)\ldots\mathbf{t}_1(k)$ is a *subsequence* of another sequence $\mathbf{t}_2 = \mathbf{t}_2(1)\mathbf{t}_2(2)\ldots\mathbf{t}_2(n)$ if and only if there exists

$q_1, q_2, \ldots, q_k$ such that $1 \leq q_1 < q_2 < q_3 \cdots < q_k \leq n$ and $\mathbf{t}_1(j) = \mathbf{t}_2(q_j)$ for all $1 \leq j \leq k$. ⌟

Definition 6.2 (Exact Manifestation of a Pattern). A pattern $\mathbf{p}$ is considered to be exactly manifested in a trace $\mathbf{t}$ at position i (for $1 \leq i \leq |\mathbf{t}|$) if and only if $\exists_{i \leq l \leq |\mathbf{t}|} \mathbf{p}$ is a subsequence of $\mathbf{t}(i, l)$ and $\forall_{i < m \leq l} \mathbf{p}$ is not a subsequence of $\mathbf{t}(m, l)$. ⌟

The second condition in the definition above restricts the consideration of a manifestation to compact contexts. In other words, if an exact manifestation of a pattern at position i subsumes another exact manifestation at position m, we make an assumption that activity executions happen with short-range dependencies and do not consider the manifestation at position i. Instead, the pattern is considered to be manifested at position m. We will later provide some ideas on relaxing this assumption in Section 6.7.

Definition 6.3 (Exact (Non-)Continuous Manifestation of a Pattern). An exact manifestation of a pattern $\mathbf{p}$ in a trace $\mathbf{t}$ at position i is said to be *continuous* if $l - i + 1 = |\mathbf{p}|$. Otherwise, it is said to be *non-continuously* manifested. ⌟

Definition 6.4 (Approximate Manifestation of a Pattern). A pattern $\mathbf{p}$ is considered to be approximately manifested in a trace $\mathbf{t}$ at position i (for $1 \leq i \leq |\mathbf{t}|$) if and only if $\mathbf{p}$ is not an exact manifestation in $\mathbf{t}$ at position i and either

- a permutation of $\mathbf{p}$ is exactly manifested in $\mathbf{t}$ at position i or
- a subsequence $\mathbf{p}'$ of $\mathbf{p}$ having a *longest common subsequence distance*[2] of $\delta \in \mathbb{N}$ with $\mathbf{p}$ (for some user defined δ) is exactly manifested in $\mathbf{t}$ at position i

The longest common subsequence condition facilitates us to capture the maximal functionality represented in $\mathbf{p}$. ⌟

[2]the longest common subsequence distance is a special case of Levenshtein edit distance, when substitutions of unlike symbols are forbidden. The permissible edit operations are exact symbol matches (substitution of like symbols), insertions, and deletions.

Algorithm 6.1 Algorithm to Preprocess a Log with Abstractions (Part 1)

```
1: Let $\mathcal{L} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n]$ be an event log defined over the set of events $\mathcal{E}$ and the set of activities $\Sigma$.
2: Let $\mathcal{M} \subseteq 2^{\Sigma} \times 2^{\Sigma^+} \times 2^{\mathcal{A}}$ be a mapping defining the abstractions
3: Let $s : \mathcal{A} \to \mathbb{L}$ be a function defined over the set of abstract activities and the sub-logs for each abstract activity; $s(\mathtt{a})$ denotes the sub-log corresponding to the abstract activity $\mathtt{a} \in \mathcal{A}$
4: Let $\widehat{PA}$ be the set of all pattern alphabets defined by $\mathcal{M}$, i.e., $\widehat{PA} = \{A \mid (A, S, X) \in \mathcal{M}\}$
5: Let $I$ be the set of all ignorable activities, i.e., $I = \Sigma \setminus \bigcup_{PA \in \widehat{PA}} PA$
6: Let $P$ be the set of patterns defined by $\mathcal{M}$, i.e., $P = \bigcup_{(A,S,X) \in \mathcal{M}} S$. In other words, $P$ defines the set of all patterns defined by the pattern alphabets involved in abstractions
7: Let $DP \subseteq \mathbb{N} \times \mathcal{E}^*$ be the set of all manifestations of patterns that needs to be disambiguated at a position in a trace
8: Let $DA \subseteq \mathbb{N} \times 2^{\mathcal{A}}$ be the set of all abstract activities that needs to be disambiguated at a position in a trace
9: Let $\mathcal{L}'$ be the transformed event log; initialize $\mathcal{L}' = [\ ]$
10: for all $\mathbf{t} \in \mathcal{L}$ do
11:     Let $\underline{\mathbf{t}}$ correspond to the sequence of activities in $\mathbf{t}$
12:     Let $\mathbf{t}' = \langle \rangle$ be the transformed trace
13:     Set $DA = \{\}$ and $DP = \{\}$
14:     for $i = 1$ to $|\underline{\mathbf{t}}|$ do
15:         if $\underline{\mathbf{t}}(i) \in I$ then
16:             continue
17:         end if
18:
19:         //Check for continuous manifestation of patterns
20:         Let $CP_i$ be the set of all patterns $\mathbf{p} \in P$ such that $\mathbf{p}$ has an exact continuous manifestation at $\underline{\mathbf{t}}(i)$
21:         Let $\hat{\mathbf{cp}} \in CP_i$ be the longest pattern manifested at $\underline{\mathbf{t}}(i)$
22:         if $CP_i = \{\}$ then $\hat{\mathbf{cp}} = \langle \rangle$
23:
24:         //Check for non-continuous manifestation of patterns
25:         Let $NP_i$ be the set of all patterns $\mathbf{p} \in P$ such that $\mathbf{p}$ has an exact non-continuous manifestation at $\underline{\mathbf{t}}(i)$
26:         Let $\hat{\mathbf{np}} \in NP_i$ be the longest pattern manifested at $\underline{\mathbf{t}}(i)$
27:         if $NP_i = \{\}$ then $\hat{\mathbf{np}} = \langle \rangle$
28:
29:         //Check for approximate manifestation of patterns
30:         Let $AP_i$ be the set of all patterns $\mathbf{p} \in P$ such that $\mathbf{p}$ has an approximate manifestation at $\underline{\mathbf{t}}(i)$
31:         Let $\hat{\mathbf{ap}} \in AP_i$ be the longest pattern manifested at $\underline{\mathbf{t}}(i)$
32:         if $AP_i = \{\}$ then $\hat{\mathbf{ap}} = \langle \rangle$
33:         Let $\hat{\mathbf{p}}$ be the longest of the patterns $\hat{\mathbf{cp}}$, $\hat{\mathbf{np}}$ and $\hat{\mathbf{ap}}$
34:         Let $X$ be the set of abstract activities to which $\hat{\mathbf{p}}$ can be associated to
```

Algorithm 6.2 Algorithm to Preprocess a Log with Abstractions (Part 2)

```
35:         **if** $X$ is a singleton, i.e., there is no ambiguity in resolving the abstraction for the pattern $\hat{\mathbf{p}}$ **then**
36:             **if** $|\hat{\mathbf{p}}| = 1$ and $\hat{\mathbf{p}} = \mathtt{a}$, $\mathtt{a} \in X$ **then**
37:                 append $\mathbf{t}(i)$ to the transformed trace, i.e., $\mathbf{t}' = \mathbf{t}' \diamond \mathbf{t}(i)$
38:             **else**
39:                 let $\hat{\mathbf{e}}$ be the sequence of events corresponding to the manifestation of $\hat{\mathbf{p}}$ in $\mathbf{t}$ at position $i$
40:                 create a new event $e$ with the activity name as the abstraction $\mathtt{a} \in X$; set the timestamp of the event $e$ to that of the last event in $\hat{\mathbf{e}}$; append the event $e$ in the transformed trace, i.e., $\mathbf{t}' = \mathbf{t}' \diamond e$
41:                 add $\hat{\mathbf{e}}$ as a new trace in the sub-log of $\mathtt{a}$, i.e., $s(\mathtt{a}) = s(\mathtt{a}) \uplus [\hat{\mathbf{e}}]$
42:             **end if**
43:         **else**
44:             mark that the pattern, abstract activity pair needs to be disambiguated later at this position in the transformed trace based on the context
45:             set $\mathbf{t}' = \mathbf{t}' \diamond \perp$; $DA = DA \cup (|\mathbf{t}'|, X)$; $DP = DP \cup (|\mathbf{t}'|, \hat{\mathbf{e}})$ where $\hat{\mathbf{e}}$ is the manifestation of $\hat{\mathbf{p}}$ in $\mathbf{t}$ at position $i$
46:         **end if**
47:         **if** $\hat{\mathbf{p}}$ is either $\widehat{\mathbf{np}}$ or $\widehat{\mathbf{ap}}$ **then** readjust the non-continuous or approximate manifestation of the pattern $\hat{\mathbf{p}}$ in both $\mathbf{t}$ and $\underline{\mathbf{t}}$
48:         Set $i = i + |\hat{\mathbf{e}}| - 1$
49:     **end for**
50:
51:     **// Disambiguate the abstractions**
52:     **for** $j = 1$ to $|\mathbf{t}'|$ **do**
53:         **if** $\mathbf{t}'(j) = \perp$ **then**
54:             Let $X$ be the set of potential abstract activities and $\hat{\mathbf{e}} \in \mathcal{E}^*$ be the manifestation corresponding to the abstraction at position $j$, i.e., $(j, X) \in DA$ and $(j, \hat{\mathbf{e}}) \in DP$
55:             **if** there exists an abstract activity $\mathtt{a} \in X$ such that $\mathtt{a}$ is in the vicinity of $j$ in $\mathbf{t}'$ **then**
56:                 create a new event $e$ with the activity name as the abstraction $\mathtt{a}$; set the timestamp of $e$ to that of the last event in $\hat{\mathbf{e}}$
57:                 Set $\mathbf{t}'(j) = e$
58:             **else**
59:                 Randomly choose an $\mathtt{a} \in X$
60:                 create a new event $e$ with the activity name as the abstraction $\mathtt{a}$; set the timestamp of $e$ to that of the last event in $\hat{\mathbf{e}}$
61:                 Set $\mathbf{t}'(j) = e$
62:             **end if**
63:             add $\hat{\mathbf{e}}$ as a new trace in the sub-log of $\mathtt{a}$, i.e., $s(\mathtt{a}) = s(\mathtt{a}) \uplus [\hat{\mathbf{e}}]$
64:         **end if**
65:     **end for**
66:     $\mathcal{L}' = \mathcal{L}' \uplus [\mathbf{t}']$
67: **end for**
```

We now look at the algorithm in detail with an example. Let us consider an event log with three traces $\mathcal{L} = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ where the simple traces (i.e., the sequences of activities) correspond to $\underline{\mathbf{t}_1}$ = `jgclfebdkelbdkahbdi`, $\underline{\mathbf{t}_2}$ = `cgjlebfdbdi`, and $\underline{\mathbf{t}_3}$ = `gclelefbd`. Let us consider the abstractions and the corresponding patterns as illustrated in Figure 6.6. Based on the mapping defined in Figure 6.6, the set of ignorable activities $I = \{\texttt{k}\}$ (Step 5, Algorithm 6.1). Algorithm 6.1 processes each trace in the event log iteratively (Step 10, Algorithm 6.1). *The algorithm scans each trace from left to right (Step 14). At any position, the algorithm checks if there exists a pattern manifestation for which an abstraction is defined. If so, the pattern manifestation in the trace is replaced with a new event signifying its abstract activity and the replaced manifestation is created as a new process instance for the sub-log corresponding to the abstract activity.*

Consider the simple trace $\underline{\mathbf{t}_1}$ = `jgclfebdkelbdkahbdi`. At position 1 (see Figure 6.7), there are two patterns, `j` and `jgc`, that are exactly continuously manifested, i.e., ($\underline{\mathbf{t}_1}(1) = \texttt{j}$ and $\underline{\mathbf{t}_1}(1,3) = \texttt{jgc}$) (Step 20, Algorithm 6.1). `jgc` is the longest of the two patterns, so $\hat{\mathbf{cp}} = \texttt{jgc}$ (Step 21, Algorithm 6.1). $NP_1 = AP_1 = \{\}$ (Steps 25–32, Algorithm 6.1). $\hat{\mathbf{p}} = \texttt{jgc}$ (Step 33, Algorithm 6.1). The abstraction corresponding to `jgc` is `W`, i.e., $X = \{\texttt{W}\}$ (Step 34, Algorithm 6.1). The sequence of events $\mathbf{e} = \langle \mathbf{t}_1(1), \mathbf{t}_1(2), \mathbf{t}_1(3)\rangle$ corresponds to the manifestation of this pattern in the trace $\mathbf{t}_1$ (Step 39). We create a new event e whose activity name is `W` to capture this abstraction. We assign the timestamp of this event to be the timestamp of $\mathbf{t}_1(3)$ (we consider the timestamp of the last event to signify the completion time of the execution of the events captured by this abstraction; however, one may also choose the timestamp of the first event to signify the start time or one may choose to have two timestamp attributes signifying the start time (timestamp of the first event) and the completion time (timestamp of the last event) of this instance of abstraction); so $\#_{\text{activity}}(e) = \texttt{W}$ and $\#_{\text{time}}(e) = \#_{\text{time}}(\mathbf{t}_1(3))$. The event e is appended to the transformed trace and a new trace $\langle \mathbf{t}_1(1), \mathbf{t}_1(2), \mathbf{t}_1(3)\rangle$ is added as a process instance to the sub-log corresponding to `W`. *For simplicity reasons, we explain this using the simple traces during our discussion.* So, in the transformed trace, we append the abstract activity `W`, i.e., $\underline{\mathbf{t}_1}' = \texttt{W}$ and add a new trace `jgc` to the sub-log corresponding to `W` (Steps 39–41, Algorithm 6.1). The position i is incremented according to the pattern, $i = 3$ (Step 48, Algorithm 6.1).

trace		transformed trace	$\mathcal{L}_{\texttt{W}}$	$\mathcal{L}_{\texttt{Y}}$	$\mathcal{L}_{\texttt{Z}}$
$\downarrow$	$CP_1 = \{\texttt{j}, \texttt{jgc}\}$				
$\overline{\texttt{j g c}}$ `l f e b d k e l b d k a h b d i`	$NP_1 = \{\}$	`W`	[`jgc`]	[ ]	[ ]
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$AP_1 = \{\}$				

Figure 6.7: Processing the patterns at position 1 with abstractions in the trace $\mathbf{t}_1$ = `jgclfebdkelbdkahbdi`.

In the next iteration, $i = 4$. At position 4 (see Figure 6.8), no exact continuous manifestation of any pattern exists, i.e., $CP_4 = \{\}$. However, an exact non-continuous manifestation of the pattern `lebd` occurs, $NP_4 = \{\texttt{lebd}\}$ and $\hat{\mathbf{np}} = \texttt{lebd}$ (Steps 25–27, Algorithm 6.1). Proceeding further, the abstraction corresponding to `lebd` is `Y`. The sequence of events $\mathbf{e} = \langle \mathtt{t}_1(4), \mathtt{t}_1(6), \mathtt{t}_1(7), \mathtt{t}_1(8) \rangle$ corresponds to the manifestation of this pattern in the trace $\mathtt{t}_1$ at position 4 (Step 39). The transformed simple trace $\underline{\mathtt{t}_1}' = \mathtt{WY}$ and the sub-log corresponding to Y, $\mathcal{L}_{\mathtt{Y}} = [\mathtt{lebd}]$ (in reality, a new event capturing the abstraction Y would be created and appended to the transformed trace and the sequence of events $\langle \mathtt{t}_1(4), \mathtt{t}_1(6), \mathtt{t}_1(7), \mathtt{t}_1(8) \rangle$ would be added as a trace in the sub-log). Step 47 requires the adjustment of the trace corresponding to the non-continuous manifestation of the pattern `lebd`. The pattern `lebd` is manifested non-continuously as $\underline{\mathtt{t}_1}(4,8) = \mathtt{lfebd}$. The adjustment refers to the shifting of activities not belonging to the pattern. In this case, the activity `f` is shifted to the end of the pattern `lebd` in the trace $\underline{\mathtt{t}_1}$. Thus, the trace $\underline{\mathtt{t}_1}$ becomes `jgclebdfkelbdkahbdi` after adjustment. The position i is incremented to $i = 7$ (Step 48, Algorithm 6.1).

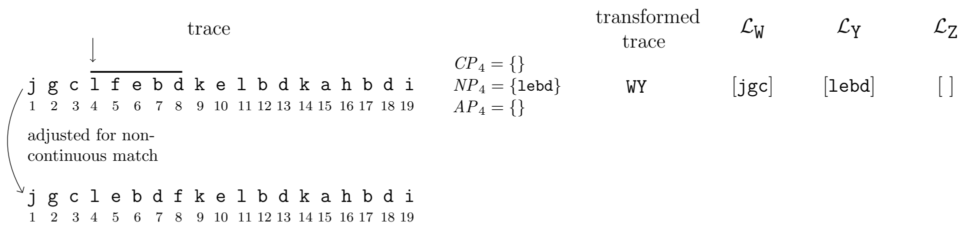

trace
transformed trace
$\mathcal{L}_{\mathtt{W}}$
$\mathcal{L}_{\mathtt{Y}}$
$\mathcal{L}_{\mathtt{Z}}$
j g c l f e b d k e l b d k a h b d i
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
$CP_4 = \{\}$
$NP_4 = \{\mathtt{lebd}\}$
$AP_4 = \{\}$
WY
[jgc]
[lebd]
[ ]
adjusted for non-continuous match
j g c l e b d f k e l b d k a h b d i
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Figure 6.8: Processing the patterns at position 4 with abstractions in the trace $\mathtt{t}_1$ = `jgclfebdkelbdkahbdi`.

In the next iteration, $i = 8$. At position 8 (see Figure 6.9), there exists an exact continuous manifestation of the pattern `f`. So, $CP_8 = \{\mathtt{f}\}$, $NP_8 = AP_8 = \{\}$. Since the abstraction corresponding to this pattern is the activity itself, we append the activity `f` to the transformed trace, i.e., $\underline{\mathtt{t}_1}' = \mathtt{WYf}$ (Steps 36–37, Algorithm 6.1) (in reality, $\mathtt{t}(8)$ would be appended to the transformed trace). No sub-log would be created for this scenario. In the next iteration, $i = 9$. At position 9 (see Figure 6.10), we have the activity $\mathtt{k} \in I$. Since this activity is specified to be ignored, we proceed for the next iteration (Step 15, Algorithm 6.1). Proceeding further and upon completion (see Figure 6.11), we get the transformed trace $\underline{\mathtt{t}_1}' = \mathtt{WYfYZi}$. The sub-logs after processing this trace correspond to $\mathcal{L}_{\mathtt{W}} = [\mathtt{jgc}]$, $\mathcal{L}_{\mathtt{Y}} = [\mathtt{lebd}, \mathtt{elbd}]$, and $\mathcal{L}_{\mathtt{Z}} = [\mathtt{ahbd}]$. Since the transformed trace does not contain any abstractions to be disambiguated, Steps 52–65 are not applicable. $\mathcal{L}' = [\mathtt{WYfYZi}]$. Figure 6.11 summarizes the transformation process for this trace.

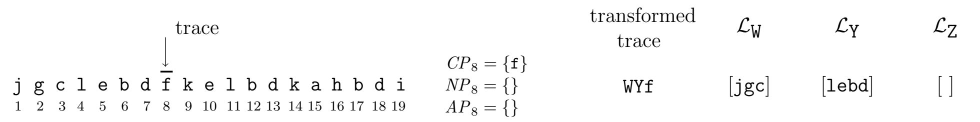


Figure 6.9: Processing the patterns at position 8 in the trace $\mathtt{t}_1$ = `jgclfebdkelbdkahbdi`.

Let us consider another trace $\underline{\mathtt{t}_2}$ = `cjglebfdbdi`. The reader is referred to Figure 6.12 for a pictorial depiction of the transformation process for this trace. At position 1, there is a continuous manifestation of the pattern `c`, $CP_1 = \{\mathtt{c}\}$ and an

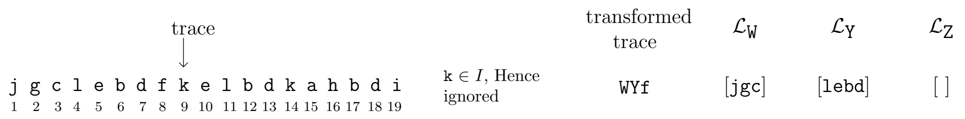


Figure 6.10: Processing the patterns at position 9 in the trace $\mathbf{t}_1$ = `jgclfebdkelbdkahbdi`.

approximate manifestation of the pattern `jgc`. The subsequence $\underline{\mathbf{t}_2}(1,3)$ = `cgj` corresponds to the permuted manifestation of the pattern `jgc`. The notion of permutation matching is discussed in [43] and efficient algorithms to detect such matches exist [43, 97]. $AP_1 = \{\texttt{jgc}\}$ and $\hat{\mathbf{ap}}$ = `jgc` (Steps 30–31, Algorithm 6.1). The longest pattern at position 1 is $\hat{\mathbf{p}}$ = `jgc` and the abstraction corresponding to it is `W`. The transformed trace $\underline{\mathbf{t}_2}'$ = `W`. The approximate manifestation of `jgc` at position 1, i.e., `cgj`, is added as a new trace to the sub-log $\mathcal{L}_W$ (Steps 39–41, Algorithm 6.1). Therefore, after Step 41, $\mathcal{L}_W = [\texttt{jgc}, \texttt{cgj}]$ (in reality, a new event capturing the abstraction is created and appended to the transformed trace and the sequence of events $\langle \mathbf{t}_2(1), \mathbf{t}_2(2), \mathbf{t}_2(3) \rangle$ is added as a new trace in the sub-log corresponding to `W`). The position i is incremented to $i = 3$.

In the next iteration $i = 4$; there exists an exact non-continuous manifestation of the pattern `lebd` at position 4, $NP_4 = \{\texttt{lebd}\}$ and $\hat{\mathbf{np}}$ = `lebd` (Steps 25–26, Algorithm 6.1). Proceeding further, the abstraction corresponding to `lebd` is `Y`. The transformed trace $\underline{\mathbf{t}_2}'$ = `WY` and the sub-log corresponding to `Y`, $\mathcal{L}_Y = [\texttt{lebd}^2, \texttt{elbd}]$. Step 47 requires the adjustment of the trace corresponding to the non-continuous manifestation of the pattern `lebd`. The pattern `lebd` is manifested non-continuously as $\underline{\mathbf{t}_2}(4,8)$ = `lebfd`; the event sequence corresponding to the manifestation is $\mathbf{e} = \langle \mathbf{t}_2(4), \mathbf{t}_2(5), \mathbf{t}_2(6), \mathbf{t}_2(8) \rangle$. The adjustment refers to the shifting of activities not belonging to the pattern. In this case, the activity `f` is shifted to the end of the pattern `lebd` in the trace $\underline{\mathbf{t}_2}$. Thus, the trace $\mathbf{t}_2$ becomes `cgjlebdfbdi` after adjustment. The position i is incremented to $i = 7$ (Step 48, Algorithm 6.1).

In the next iteration, $i = 8$. At position 8, there exists an exact continuous manifestation of the pattern `f`. So, $CP_8 = \{\texttt{f}\}$, $NP_8 = AP_8 = \{\}$. Since the abstraction corresponding to this pattern is the activity itself, we append the activity `f` to the transformed trace, i.e., $\underline{\mathbf{t}_2}'$ = `WYf` (Steps 36–37, Algorithm 6.1). In the next iteration, $i = 9$. At position 9, we have the exact continuous manifestation of `bd`. So, $CP_9 = \{\texttt{bd}\}$, $NP_9 = AP_9 = \{\}$. $\hat{\mathbf{p}}$ = `bd` and there are two abstractions, `Y` and `Z`, corresponding to `bd` (Steps 44–45, Algorithm 6.1). At this moment, we cannot disambiguate this. We postpone it until the entire trace is processed. The transformed trace $\underline{\mathbf{t}_2}' = \texttt{WYf}\perp$, $DA = \{(4, \{\texttt{Y}, \texttt{Z}\})\}$ and $DP = \{(4, \langle \mathbf{t}_2(9), \mathbf{t}_2(10) \rangle)\}$ (Step 45, Algorithm 6.1). The position i is incremented to $i = 10$. In the next iteration, $i = 11$ and there exists an exact continuous manifestation of the pattern `i`. Proceeding further, we get $\underline{\mathbf{t}_2}' = \texttt{WYf}\perp\texttt{i}$.

Now, the unresolved abstract activities in the transformed trace need to be disambiguated (Step 51, Algorithm 6.1). The abstract activity at position 4 in the transformed trace needs to be disambiguated. We have an abstract activity $\texttt{Y} \in \{\texttt{Y}, \texttt{Z}\}$

Iter	Index i	trace		transformed trace	$\mathcal{L}_W$	sub logs $\mathcal{L}_Y$	$\mathcal{L}_Z$
1	1	$\downarrow\overline{\texttt{j g c}}$ `l f e b d k e l b d k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_1 = \{\texttt{j}, \texttt{jgc}\}$ $NP_1 = \{\}$ $AP_1 = \{\}$	`W`	[`jgc`]	[ ]	[ ]
2	4	`j g c` $\downarrow\overline{\texttt{l f e b d}}$ `k e l b d k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_4 = \{\}$ $NP_4 = \{\texttt{lebd}\}$ $AP_4 = \{\}$	`WY`	[`jgc`]	[`lebd`]	[ ]
adjusted for non-continuous match							
3	8	`j g c l e b d` $\downarrow\overline{\texttt{f}}$ `k e l b d k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_8 = \{\texttt{f}\}$ $NP_8 = \{\}$ $AP_8 = \{\}$	`WYf`	[`jgc`]	[`lebd`]	[ ]
4	9	`j g c l e b d f` $\downarrow$`k e l b d k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	`k` $\in I$, Hence ignored	`WYf`	[`jgc`]	[`lebd`]	[ ]
5	10	`j g c l e b d f k` $\downarrow\overline{\texttt{e l b d}}$ `k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_{10} = \{\texttt{elbd}\}$ $NP_{10} = \{\}$ $AP_{10} = \{\}$	`WYfY`	[`lebd`, `elbd`]		[ ]
6	14	`j g c l e b d f k e l b d` $\downarrow$`k a h b d i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	`k` $\in I$, Hence ignored	`WYfY`	[`jgc`]	[`lebd`, `elbd`]	[ ]
7	15	`j g c l e b d f k e l b d k` $\downarrow\overline{\texttt{a h b d}}$ `i` 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_{15} = \{\texttt{ahbd}\}$ $NP_{15} = \{\}$ $AP_{15} = \{\}$	`WYfYZ`	[`jgc`]	[`lebd`, `elbd`]	[`ahbd`]
8	19	`j g c l e b d f k e l b d k a h b d` $\downarrow\overline{\texttt{i}}$ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19	$CP_{19} = \{\texttt{i}\}$ $NP_{19} = \{\}$ $AP_{19} = \{\}$	`WYfYZi`	[`jgc`]	[`lebd`, `elbd`]	[`ahbd`]

Figure 6.11: Transforming the trace $\underline{\texttt{t}_1}$ = `jgclfebdkelbdkahbdi` with abstractions.

at position 2 (Step 55, Algorithm 6.1). We assume that the activity at position 4 is in the same context as in 2 and incline towards disambiguating it to `Y`. In general, we prefer the closest abstraction in the vicinity for disambiguation. We set $\underline{\mathbf{t}_2}'(4) = \mathtt{Y}$ and add the manifestation `bd` as a new trace to the sub-log corresponding to `Y`. Thus, $\underline{\mathbf{t}_2}' = \mathtt{WYfYi}$ and $\mathcal{L}_{\mathtt{Y}} = [\mathtt{lebd}^2, \mathtt{elbd}, \mathtt{bd}]$. $\mathcal{L}' = [\mathtt{WYfYYi}, \mathtt{WYfYi}]$ after processing of the second trace.

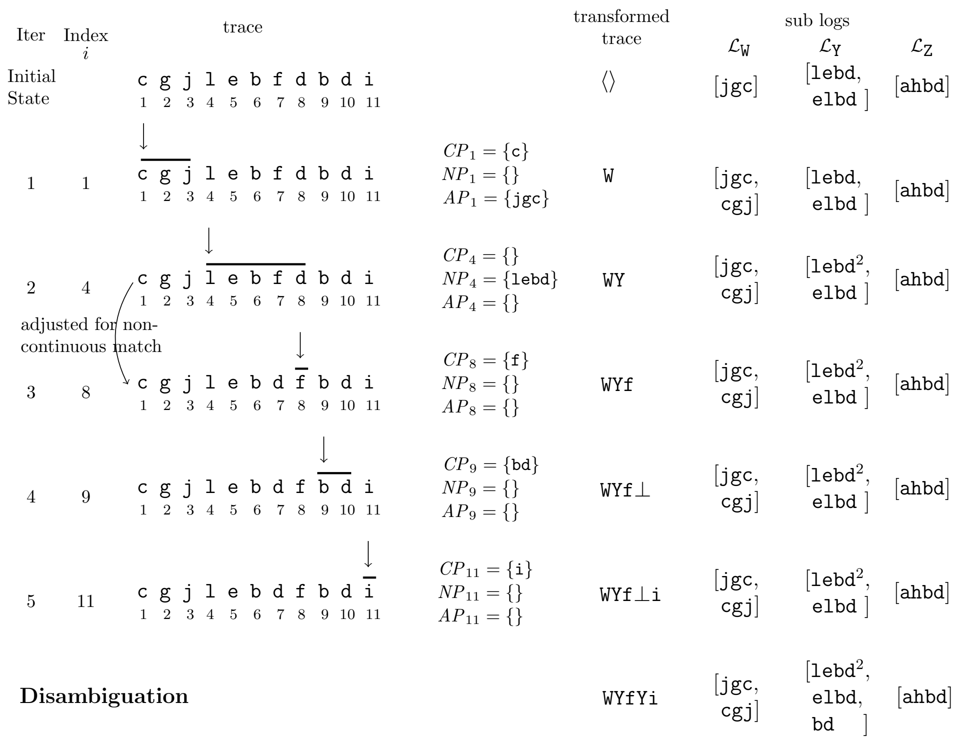


Figure 6.12: Transforming the trace $\underline{\mathbf{t}_2} = \mathtt{cgjlebfdbdi}$ with abstractions.

Let us consider the third trace $\underline{\mathbf{t}_3} = \mathtt{gclelefbd}$. The reader is referred to Figure 6.13 for a pictorial depiction of the transformation process for this trace. As position 1, there is an exact continuous manifestation of the pattern `g`. There is also an approximate manifestation of the pattern `jgc` (where the activity `j` is skipped). The longest common subsequence distance between the manifestation `gc` and the pattern `jgc` is 1. $\hat{\mathbf{p}} = \mathtt{jgc}$ and the transformed trace $\underline{\mathbf{t}_3}' = \mathtt{W}$. We add a new trace `gc` corresponding to the approximate manifestation of the pattern `jgc` to the sub-log of this abstraction $\mathcal{L}_{\mathtt{W}}$ (Step 41, Algorithm 6.1). In the next iteration, $i = 3$. The definition of the exact non-continuous manifestation of a pattern prohibits the consideration of the subsequence $\underline{\mathbf{t}_3}(3,9) = \mathtt{lelefbd}$ as the manifestation for the pattern `lebd`. This is due to the fact that there exists a subsuming sequence $\underline{\mathbf{t}_3}(5,9) = \mathtt{lefbd}$ that qualifies to be a non-continuous manifestation of the pattern `lebd`. Hence, we consider the exact continuous manifestation of the pattern `le` at position 3 whose abstraction is `Y`. Proceeding further, the transformed trace for this trace would be $\underline{\mathbf{t}_3}' =$ `WYYf`. After the processing of the three traces, the transformed log $\mathcal{L}' =$ [`WYfYZi`, `WYfYi`, `WYYf`] and the sub-logs for the abstract activities correspond to $\mathcal{L}_{\mathtt{W}} =$ [`jgc`, `cgj`, `gc`], $\mathcal{L}_{\mathtt{Y}} =$ [`lebd`^{3}, `elbd`, `bd`, `le`], and $\mathcal{L}_{\mathtt{Z}} =$ [`ahbd`].

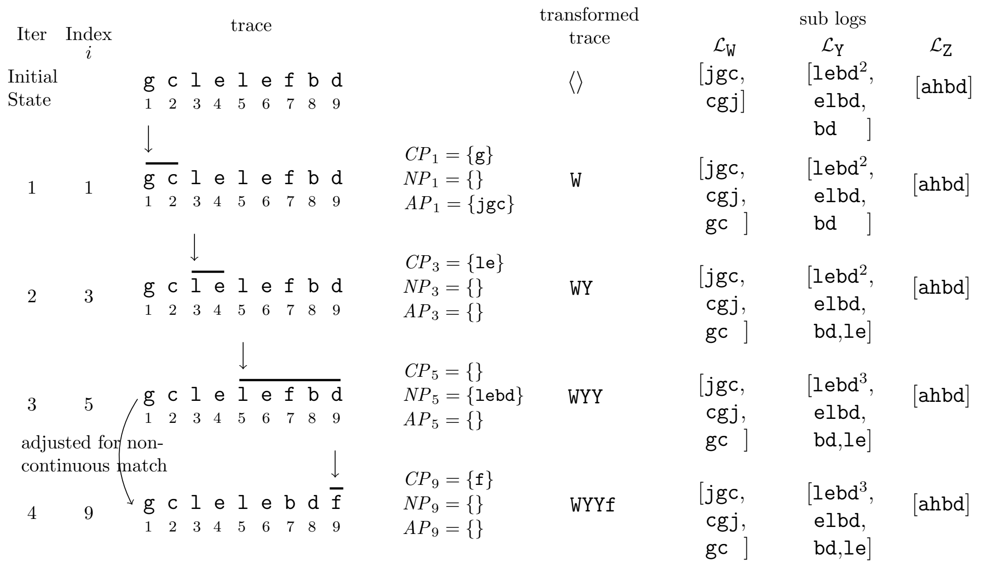


Figure 6.13: Transforming the trace $\underline{\mathbf{t}_3}$ = `gclelefbd` with abstractions.

6.4 Discovering Processes at a Particular Level

In the previous section, we have seen how event logs containing low-level events can be transformed to higher levels based on abstractions defined over low-level events. In this section, we show how such a transformed log can be used to mine processes at a particular level. *We hypothesize that our transformation approach assists in improving firstly, the comprehensibility, and secondly, the quality of the mined models.* We illustrate this with a small example. Figure 6.14 depicts a process model in YAWL with 11 activities, $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{l}\}$. Note that activities `b` and `d` can appear in different contexts: in a loop with `l` and `e` as alternatives and preceded by `a` and `h`. We have modeled the process model depicted in Figure 6.14 in CPN tools [180] and generated event logs using simulation. We use one event log containing 75 traces, 11 event classes, and 1047 events for our discussion.

Figure 6.15 depicts the process model mined using the α++ algorithm [51][3]. Since the simulated event logs correspond to a well structured model, we chose the α++ algorithm as the starting point. Many other discovery algorithms were also studied

[3]An artificial start and end task is added to all the traces in the log before discovering the models.

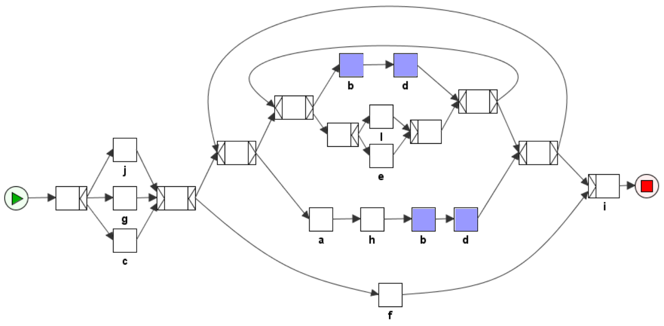


Figure 6.14: A simple YAWL process model.

(cf. Table 6.2) but we first discuss the results from α++ algorithm. We can see from Figure 6.15 that this event log, though generated from a simple process, resulted in a spaghetti-model. The spaghettiness can be attributed to the difficulty of the α++ algorithm in dealing with concurrency and duplicate tasks. Among the three parallelism constructs in the process model (one involving the activities `j`, `g`, and `c` at the beginning of the process model, one involving activities `l` and `e`, and one involving the activity `f`), the execution of activity `f` is the most challenging one for the α++ algorithm. Note that `f` can execute in parallel to another fragment that involves a loop, XOR, and AND construct. Another issue stems from the presence of duplicate tasks `b` and `d`. α++ algorithm is not capable of dealing with duplicate tasks; it considers them to be the same task. In spite of its spaghettiness, this model has a fitness of 0.91 (we use the conformance checker plug-in in ProM 5.2 to measure the fitness [191] values).

We use the common execution patterns manifested in the event log and define abstractions using the concepts presented in Chapter 3. Figure 6.6 depicts the pattern graph obtained from this event log. From the pattern graph, we select the mapping defined as in Figure 6.6 for abstractions. The event log is transformed with these abstractions using the algorithm discussed in Section 6.3. The transformation resulted in a transformed log, $\mathcal{L}'$, with 75 traces, 5 event classes, and 408 events and three sub-logs $\mathcal{L}_{\mathtt{W}}$, $\mathcal{L}_{\mathtt{Y}}$, and $\mathcal{L}_{\mathtt{Z}}$ corresponding to the abstractions `W`, `Y`, and `Z` respectively. Figure 6.16 depicts the first-level process model discovered using the α++ algorithm on this transformed log. The fitness of this model is 1.0. We can see that by transforming the log with abstractions defined over common execution patterns, we are able to discover a model that is well-structured and comprehensible.

We can use the sub-logs created for each abstract activity to mine the processes corresponding to them. Figure 6.17 depicts the subprocess mined using the α++ algorithm for the abstract activity `W` while Figures 6.18 and 6.19 depict the sub-processes mined for the abstract activities `Y` and `Z` respectively. Thus we generated a process model with one level of hierarchy. Abstractions can further be defined on the transformed log $\mathcal{L}'$. Table 6.1 depicts the patterns and the corresponding

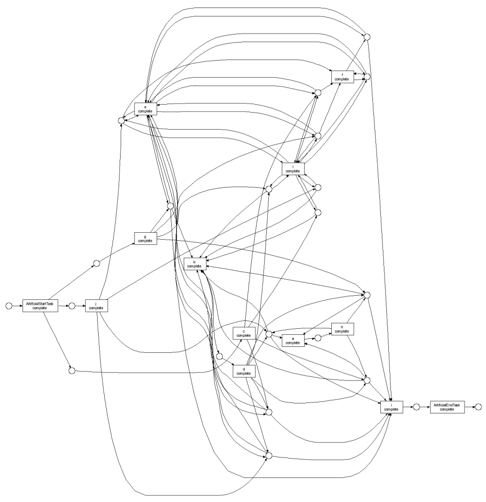


Figure 6.15: The process model mined on the raw event log using the α++ algorithm.

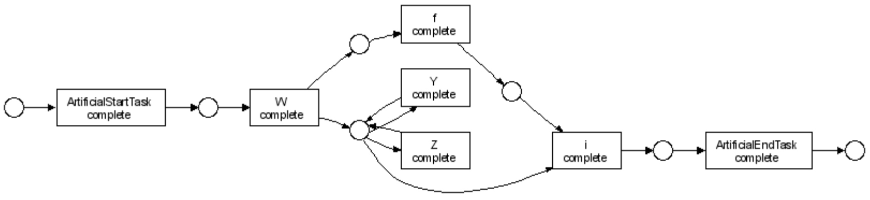


Figure 6.16: The process model mined using the α++ algorithm on the log obtained after one level of transformation with abstractions. The activities `W`, `Y`, and `Z` are abstract activities.

abstractions defined over this transformed log. We have now defined an abstraction over the activities that are abstractions themselves defined in the previous level. This creates a second level of hierarchy. Upon preprocessing the log with these abstractions, we obtain a transformed log $\mathcal{L}''$ that contains 75 traces, 4 event classes, and 394 events and a sub-log $\mathcal{L}_{\mathtt{M}}$. Figure 6.20(a) depicts the process model mined using the α++ algorithm on the transformed log $\mathcal{L}''$. Figure 6.20(b) depicts the process model mined using the sub-log generated for the abstract activity `M`. The abstract activities `Y` and `Z` in this model correspond to the abstractions defined in the previous iteration. Note the similarity of this model with that of the original model in Figure 6.21. Figure 6.21 depicts the same YAWL model (as in Figure 6.14) but annotated with regions that have been abstracted. Using the common execution patterns, we are able to identify meaningful process fragments as abstractions.

In this fashion, one can exploit the abstractions of low level events to transform a log to an appropriate level and use the abstracted log to mine the processes. Furthermore, the sub-logs created for each abstraction can be used to uncover the subprocesses captured by these abstractions. Clearly, we can see that the models

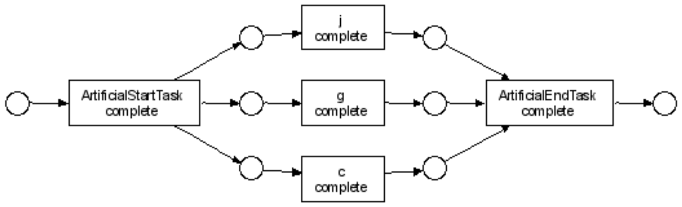


Figure 6.17: The process model, depicting the subprocess for the abstraction W, mined using the α++ algorithm on the sub-log generated during the transformation process for the abstract activity W.

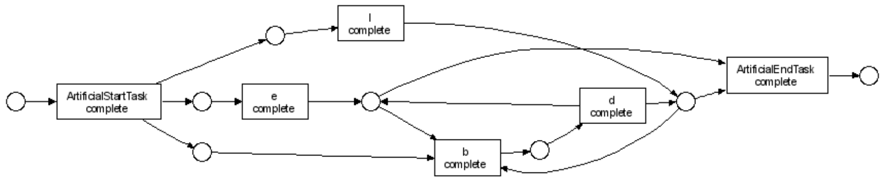


Figure 6.18: The process model, depicting the subprocess for the abstraction Y, mined using the α++ algorithm on the sub-log generated during the transformation process for the abstract activity Y.

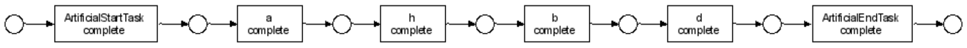
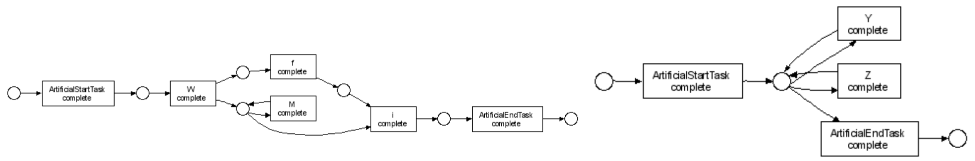


Figure 6.19: The process model, depicting the subprocess for the abstraction Z, mined using the α++ algorithm on the sub-log generated during the transformation process for the abstract activity Z.

Table 6.1: Pattern alphabets and abstractions.

Pattern Alphabet	Patterns	Abstraction
{Y, Z}	{ZY, YZ}	M
{Y}	{Y}	M
{Z}	{Z}	M
{W}	{W}	W
{f}	{f}	f
{i}	{i}	i


ArtificialStartTask complete
W complete
f complete
M complete
i complete
ArtificialEndTask complete
ArtificialStartTask complete
Y complete
Z complete
ArtificialEndTask complete

(a) top-level model (b) subprocess for M

Figure 6.20: The top-level process model mined using the α++ algorithm on the log obtained after two levels of transformation with abstractions, and the subprocess for the abstract activity M.

mined using the two-phase approach are much simpler and easily comprehensible.

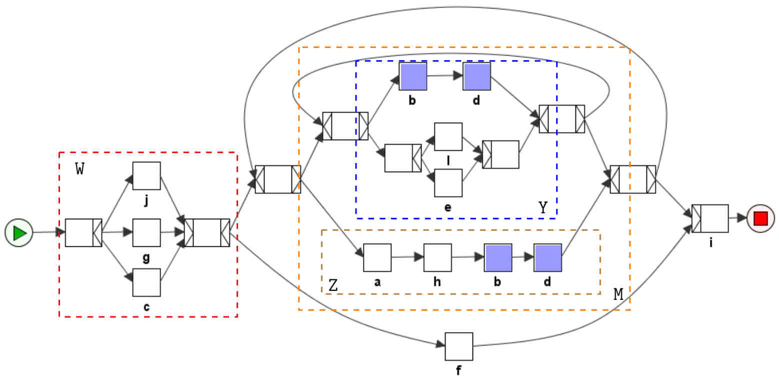


Figure 6.21: A simple YAWL process model with marked regions that represent process fragments that are abstracted.

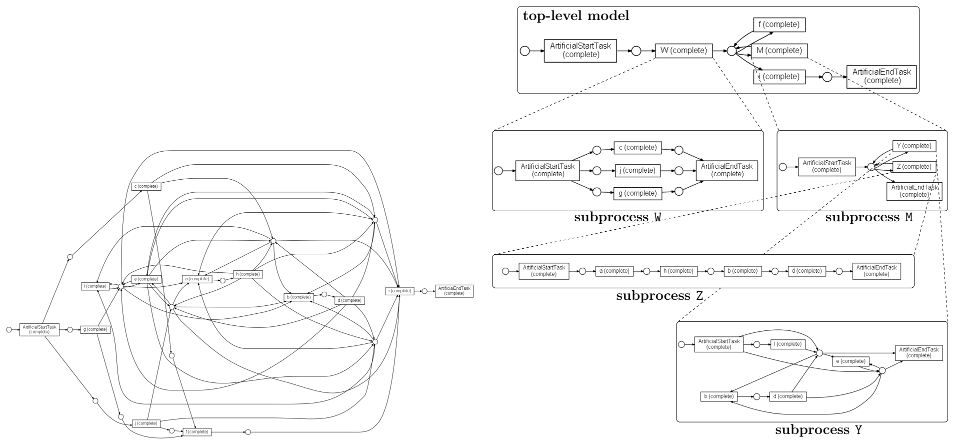
It is important to note that any process discovery algorithm can be used to mine the (sub)-processes on the transformed log and the sub-logs. Table 6.2 depicts the fitness measures of the process models mined using various process discovery algorithms based on the traditional approach, and the two-phase approach (with abstractions). We considered the discovery algorithms that can mine a Petri net from the event log. The heuristic nets generated by the Heuristics Miner [264], Genetic Miner [53, 231], and Genetic Miner with duplicate tasks [49] are converted to Petri nets before measuring for conformance[4]. The 'Raw Model' corresponds to the model mined on the event log without any abstractions. Since the two-phase approach generates multiple models (the top-level model and models for the different abstractions), we consider the average fitness of the different models as a reflection of the overall quality. As shown in Table 6.2, the two-phase approach enables the discovery of models that exhibit a better conformance (fitness) when compared to the traditional approach irrespective of the discovery algorithm adopted. Furthermore, the mined models are less complex and easily comprehensible. The ILP Miner [238] can mine a model that guarantees a fitness of 1.0 for all event logs. Thus we see a fitness of 1.0 for the model mined on the raw event log. Figure 6.22(a) depicts the process model obtained on the raw event log using the ILP miner. As is evident, this model is quite spaghetti-like. In contrast, Figure 6.22(b) depicts the process map obtained using our two-phase approach. We can clearly see that the process map is much simpler and more comprehensible.

Ideally, the evaluation of any process discovery approach should involve four competing quality dimensions [167], viz., *fitness*, *simplicity*, *precision*, and *generalization*. Fitness measures the degree to which a mined model can explain the behavior observed in the event log. Precision and generalization measures the degree of *un-*

[4]For the genetic miner variants, we have fixed the number of generations to 100. The rest of the parameters (e.g., the number of individuals (population size), crossover type/rate, mutation type/rate, etc.) are set to their default values.

Table 6.2: Comparison between the quality (using fitness [191] as a measure) of the mined models using the traditional approach and the two-phase approach (with abstractions defined over common execution patterns). The same discovery algorithm is used across the entire row.

Discovery Algorithm	Traditional	Two-phase Approach					
	Approach						
	Raw Model	Top Level	W	M	Y	Z	Avg
α	0.82	1.00	1.00	1.00	0.83	1.00	**0.966**
α++	0.91	1.00	1.00	1.00	0.96	1.00	**0.992**
Heuristics Miner	0.93	1.00	1.00	0.99	0.95	1.00	**0.988**
Genetic Miner	0.80	1.00	1.00	1.00	1.00	1.00	**1.000**
Genetic Miner (with duplicate tasks)	0.90	1.00	1.00	1.00	0.99	1.00	**0.998**
Region miner	1.00	1.00	1.00	1.00	0.95	1.00	0.990
ILP Miner	1.00	1.00	1.00	1.00	1.00	1.00	**1.000**



(a) traditional approach (b) our two-phase approach

Figure 6.22: The process model obtained on the raw event log and the process map obtained by our two-phase approach using the ILP miner as the discovery algorithm. The subprocesses corresponding to the abstract activities in the process map are depicted as a tree of models.

derfitting and *overfitting* while spaghattiness/comprehensibility acts as a measure of simplicity of the mined process models. Any discovery algorithm that strikes a balance between all of these dimensions emerges as a winner. However, this turns out to be a challenging problem. In this thesis, we assess our techniques using only the fitness and simplicity dimensions.

In order to have a seamless zoom-in and zoom-out facility (on the abstract activities), the implementation of process discovery algorithms need to be aware of the availability of the sub-log for each abstract activity, which can be used to mine and depict a model upon zooming in on an abstract activity. We have extended the Fuzzy miner's [94] implementation in ProM 6.0 with this feature. We call this extension, the Fuzzy Map Miner (cf. Chapter 10). Fuzzy models are discovered for each of the sub-logs and are displayed upon zooming in on its corresponding abstract activity. Abstract activities are differentiated from other activities by means of a distinct color (a darker shade of blue, see also Figure 6.23). Figure 6.23 depicts the process model mined on the transformed log $\mathcal{L}''$, which is at two-levels of hierarchy, along with the tree of process models for the hierarchy of abstractions. The blue (dark) colored nodes in the top-level model in Figure 6.23, corresponding to the activities `W` and `M`, are abstract activities that can be zoomed in. Upon zooming in on the abstract activity `W`, the model depicted in Figure 6.23 corresponding to the subprocess `W` is shown (the sub-log $\mathcal{L}_{\mathtt{W}}$ is used to discover this model). Likewise, the process models for the abstract activities `M`, `Y`, and `Z` can be shown upon zooming into the respective abstract activities as depicted in Figure 6.23.

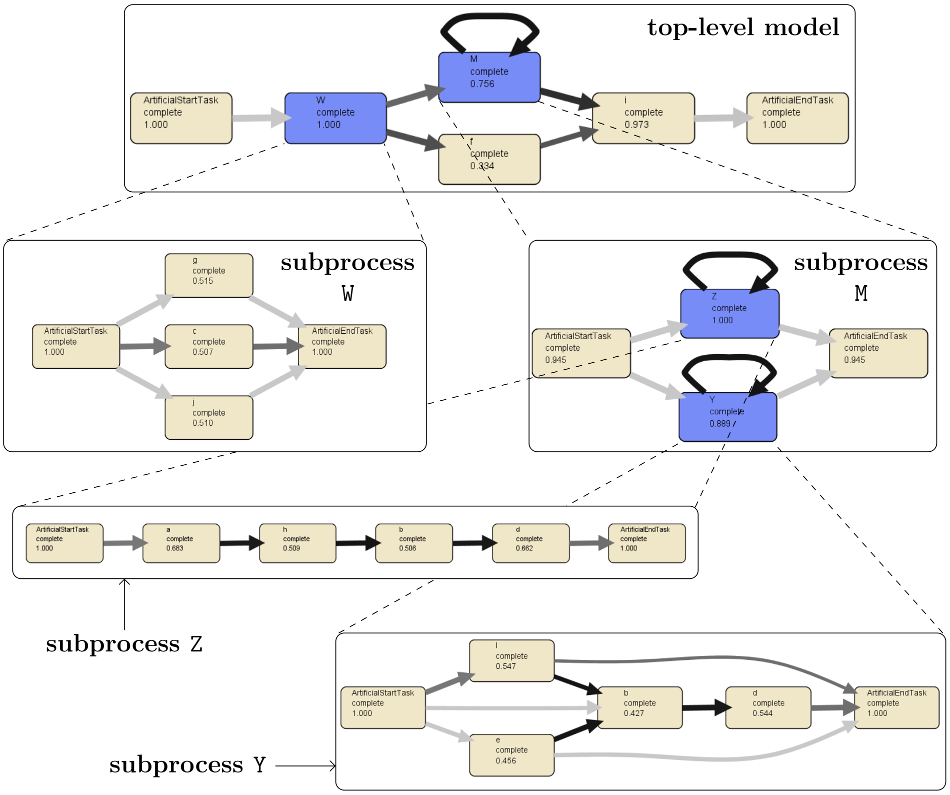


Figure 6.23: The top-level fuzzy model and the tree of fuzzy models corresponding to the hierarchy of abstractions obtained using our two-phase approach.

Thus, by selecting relevant and context-dependent abstractions, we can transform a low-level event log to an event log containing higher levels of abstraction. The transformed log along with the sub-logs generated for each abstract activity can be used to create a hierarchical process map with multiple levels of hierarchy. We can use this approach to create maps that are simple (less complex and more comprehensible) and of high quality. A process map with hierarchies can be visualized as a tree as depicted in Figure 6.24. Each node in the tree represents a process model in the process map and the depth of a node indicates the level of the corresponding process model in the process map. The tree depicted in Figure 6.24 corresponds to the process map depicted in Figure 6.23. In addition, the nodes signifying subprocesses in the process map are annotated with the abstract activity they correspond to.

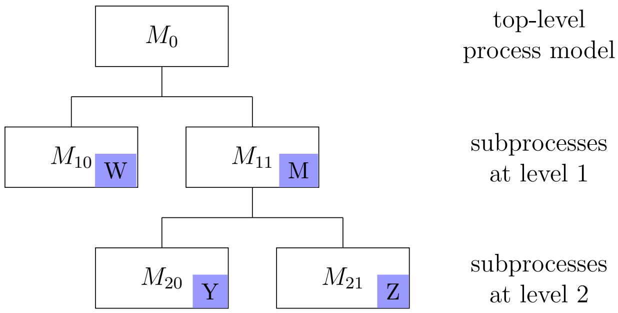


Figure 6.24: A process map visualized as a tree of process models. Subprocesses are annotated with the abstract activity they correspond to. For example, the subprocess M_{10} corresponds to the abstract activity `W`, which is a node in M_0.

6.5 Experiments and Discussion

In this section, we demonstrate the discovery of process maps for the digital copier example. We have considered a log $\mathcal{L}$ containing 100 traces, 76 event classes, and 40,995 events. As discussed in Chapter 3, we recommend the definition of abstractions over tandem arrays first (recall that tandem arrays capture loop constructs). Table 6.3 depicts the pattern alphabets and the abstractions defined for them. Pattern alphabet 1 corresponds to the loop construct in the Capture Image subprocess of the digital copier example (cf. Figure 3.2 in Chapter 3). Pattern alphabet 2 captures the Half Toning subprocess (cf. Figure 3.5 in Chapter 3), which is modeled as a loop construct, in its entirety as a tandem array. Pattern alphabet 3 corresponds to the loop construct within the Writing subprocess (cf. Figure 3.7 in Chapter 3). Pattern alphabets 4 and 5 correspond to the loop construct within the Developing subprocess (cf. Figure 3.8 in Chapter 3). This loop construct involves a XOR construct and the two pattern alphabets correspond to the two branches of the XOR construct. Pattern alphabets 6 and 7 correspond to the Rasterize Image subprocess (cf. Figure 3.3 in Chapter 3). The process fragment involving these activities is not explicitly modeled as a loop construct. However, we notice a tandem manifestation of these activities. This is attributed to the following reason: for jobs that are print requests, each page of the document is first interpreted and then rendered. The interpretation and rendering can proceed simultaneously after the interpretation of the first page. The interpretation of pages is relatively faster than rendering. Once all pages in a document are interpreted, it is only the rendering process that needs to be completed and this manifests as a tandem array. Pattern alphabets that have a lot of commonalities are merged together and abstractions are defined over the merged alphabet, e.g., pattern alphabets 4 and 5, and 6 and 7 in Table 6.3.

Table 6.3: Pattern alphabets and abstractions defined using the tandem array patterns on the raw event log $\mathcal{L}$.

S.No	Pattern Alphabet	Abstraction
1	{Illuminate Page, Move Scan Head, Focus Light Beam, A/D Conversion, Interpolate, Filtered Image, Collect Image}	Capture Image
2	{Error Diffusion Method Start, Load Quantization Pixel, Neighbor Quantization Error Packing, Calculate Total Neighbor Quantization Error, Subtract, Table Based Multi-level Quantizer, Store Quantization Pixel, Calculate Quantization Error, Error Diffusion Method Complete }	Half Toning
3	{Emit Laser, Photons Travel to Drum, Reverse Charges}	Charge Drum
4	{Drum Spin Start, Coat Toner on Drum, Drum Spin Stop}	Coat Toner
5	{Drum Spin Start, Coat Light Toner on Drum, Drum Spin Stop}	
6	{Rendering, Screening Start, FM Screening, Screening Complete, Current Page Image, Accumulate Images}	Render and Screen
7	{Rendering, Screening Start, FM Screening, Screening Complete, Current Page Image, Accumulate Images}	

The event log is transformed based on the procedure explained in Section 6.3 using the abstractions and pattern alphabets (along with the patterns captured by these pattern alphabets) defined as in Table 6.3. The abstracted log $\mathcal{L}'$ contains 100 traces, 53 event classes and $27,060$ events. In addition, five sub-logs corresponding to each of the five abstractions defined in Table 6.3 are generated. The abstract activities thus formed are at the first level of hierarchy. Repetitive application of this process, viz., defining abstractions and transforming the log, enables the discovery of multiple levels of hierarchy (new abstractions can be defined over existing abstractions). The transformed log obtained in iteration i is used as the input event log in iteration $i + 1$. We considered the maximal repeat patterns in the transformed log $\mathcal{L}'$ and defined the abstractions as depicted in Table 6.4. Pattern alphabet 1 corresponds to the Image Processing subprocess (cf. Figure 3.4 in Chapter 3). Pattern alphabet 1 in addition contains the activity 'Store Image' that follows Image Processing (cf. Figure 3.1 in Chapter 3). Note that this pattern alphabet contains the activity Half Toning, which is an abstract activity defined in the previous iteration. Pattern alphabet 2 corresponds to the loop construct involving the activities 'Developing', 'Fusing', and 'Wipe Toner on Drum' in the Print Image subprocess (cf. Figure 3.6 in Chapter 3). This pattern alphabet also contains an activity, Coat Toner, resulting from an abstraction defined in the previous iteration. Thus the level of abstraction (hierarchy) is two. The event log $\mathcal{L}'$ is transformed using the abstractions defined in Table 6.4. This results in an abstracted log $\mathcal{L}''$ and two sub-logs. The abstracted log $\mathcal{L}''$ contains 100 traces, 31 event classes and $7,036$ events.

Table 6.4: Pattern alphabets and abstractions defined using the maximal repeat patterns on the event log $\mathcal{L}'$.

S.No	Pattern Alphabet	Abstraction
1	{Scanner Compensation, Photo Quality Reproduction, Zooming Start, X-Zoom, Y-Zoom, Zooming Complete, Half Toning, Rotate, Overlay, Compression, Store Image}	Image Processing
2	{Coat Toner, Drum Spin Start, Paper Roller Spin Start, Transfer Toner, Paper Roller Spin Stop, Drum Spin Stop, Fusing Start, Heated Roller Spin Start, Apply Heat, Heated Roller Spin Stop, Pressure Roller Spin Start, Apply Pressure, Pressure Roller Spin Stop, Fusing Complete, Wipe Toner on Drum}	Developing and Fusing

We considered the event log $\mathcal{L}''$ for further abstractions. Table 6.5 depicts the pattern alphabets and the abstractions thus formed using the maximal repeat patterns. Pattern alphabets 1, 2, and 3 correspond to the interpretation of pages for print job requests in the Rasterize Image subprocess (cf. Figure 3.3 in Chapter 3). There are three types of pages that can be interpreted: (a) post script, (b) unformatted text, and (c) page control language. This is modeled using an XOR construct. The three pattern alphabets correspond to the interpretation of each type of the page. Since the patterns are similar, we merge them and define an abstraction for the merged alphabet. Pattern alphabet 4 corresponds to the Print Image subprocess (cf. Figure 3.6 in Chapter 3). This pattern alphabet contains the abstract activity Developing and Fusing, which is at an abstraction level of two. Thus the abstract activity Print Image is at a hierarchy level of three. We transform the event log using these abstractions. The resulting abstract log $\mathcal{L}'''$ contains 100 traces, 18 event classes, and $1,213$ events. In addition, two sub-logs corresponding to the two abstractions are generated.

Table 6.5: Pattern alphabets and abstractions defined using the maximal repeat patterns on the event log $\mathcal{L}''$.

S.No	Pattern Alphabet	Abstraction
1	{Interpretation Start, Unformatted Text, Interpretation Complete}	Interpret
2	{Interpretation Start, Post Script , Interpretation Complete}	
3	{Interpretation Start, Page Control Language, Interpretation Complete}	
4	{Writing Start, Drum Spin Start, Read Image, Apply Negative Charge on Drum, Charge Drum, Drum Spin Stop, Image Created on Drum, Writing Complete, Developing and Fusing, Erase Charge on Drum }	Print Image

Having transformed the log to a desired level of granularity, we now mine the process maps. Figure 6.25 depicts the top-level model mined using the Fuzzy Map miner on the abstract log $\mathcal{L}'''$. The nodes colored in blue (dark colored) are the abstract nodes corresponding to the abstract activities. Again, this model is in tune with the original model that we used to simulate the logs (cf. Figure 3.1 in Chapter 3). Abstract nodes can be seamlessly zoomed in to view the subprocesses underneath it. Figure 6.25 also depicts three of the subprocesses corresponding to the abstract activities Interpret, Render and Screen, and Capture Image in the top-level model. Figure 6.26 depicts the fuzzy model mined for Image Processing. Note that this model is in tune with that of the original model (cf. Figure 3.4 in Chapter 3). This fuzzy model contains an abstract activity Half Toning. Upon zooming on this abstract node, we obtain the fuzzy model as depicted in Figure 6.26. Figure 6.27 depicts the fuzzy model for the abstract activity Print Image and the tree of fuzzy models corresponding to the abstract activities within this subprocess. The process fragment between 'Writing Start' and 'Writing Complete' nodes (both inclusive) in this fuzzy model corresponds to the Writing subprocess of the original model (cf. Figure 3.7 in Chapter 3). This process fragment contains an abstract activity Charge Drum whose corresponding fuzzy model is as depicted in Figure 6.27. The Print Image fuzzy model contains another abstract activity Developing and Fusing, the fuzzy model of which is also depicted in Figure 6.27. The process fragment between 'Coat Toner' and 'Drum Spin Stop' (both inclusive) corresponds to the Developing subprocess of the original model (cf. Figure 3.8 in Chapter 3) while the process fragment between 'Fusing Start' and 'Fusing Complete' corresponds to the Fusing subprocess of the original model (cf. Figure 3.9 in Chapter 3). The fuzzy model corresponding to Developing and Fusing contains an abstract activity Coat Toner, the fuzzy model of which is depicted in Figure 6.27.

In this fashion one can mine process maps. We have demonstrated the discovery of process maps using the Fuzzy Map miner. However, as discussed earlier, one can use any process discovery algorithm. Abstract activities can be seamlessly zoomed in/out by depicting the processes mined using the sub-log corresponding to that abstract activity. The approach proposed in Chapter 3 to form abstractions by exploiting common execution patterns shows significant promise in defining domain significant abstractions (as is evident in the digital copier example). Furthermore, the transformation approach presented in this chapter enables the preprocessing of event logs with abstractions and generates sub-logs for each of the abstractions.

Let us compare this with Figure 6.28(a), which corresponds to the model uncovered using the classical Fuzzy miner [94] (with default settings) on the raw event log. As discussed in Section 6.1, the Fuzzy miner groups activities into cluster nodes (blue (dark) colored nodes in Figure 6.28(a)) based on their significance/correlation. Figure 6.28(b) depicts the process model obtained upon zooming in on one of the cluster nodes containing 55 activities. As we can see, the model is spaghetti-like and hard to comprehend. The cluster nodes formed by the classical Fuzzy miner do not have any semantic significance with respect to the domain/application. In other words, the classical Fuzzy miner poses the risk of aggregating unrelated nodes together in a cluster (using the map metaphor, this is similar to streets in Eindhoven being clus-

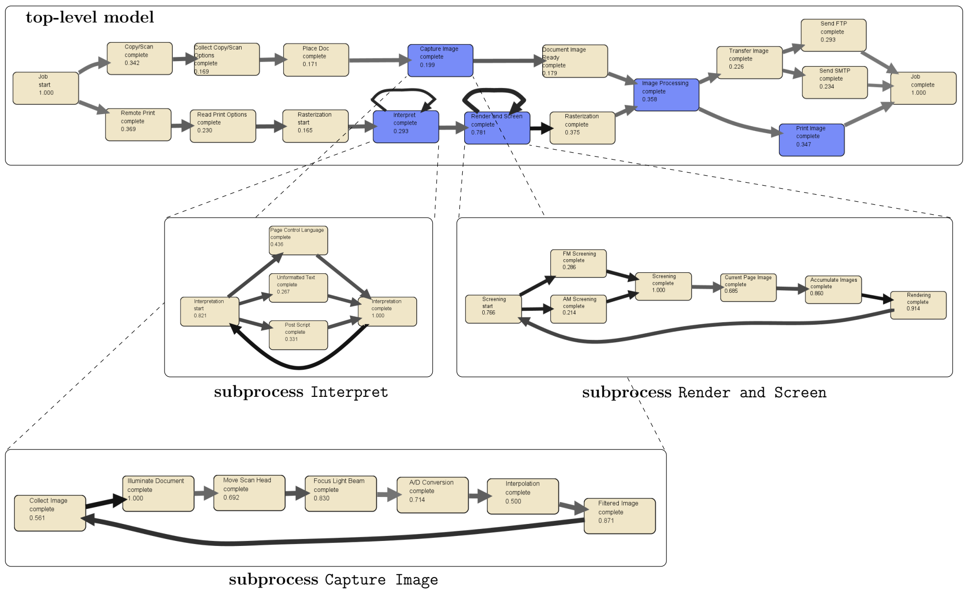

top-level model
Job start 1.000
Copy/Scan complete 0.342
Collect Copy/Scan Options complete 0.169
Place Doc complete 0.171
Capture Image complete 0.199
Document Image Ready complete 0.179
Image Processing complete 0.358
Transfer Image complete 0.226
Send FTP complete 0.293
Send SMTP complete 0.234
Job complete 1.000
Remote Print complete 0.369
Read Print Options complete 0.230
Rasterization start 0.165
Interpret complete 0.293
Render and Screen complete 0.781
Rasterization complete 0.375
Print Image complete 0.347
Page Control Language complete 0.436
Unformatted Text complete 0.267
Interpretation start 0.821
Interpretation complete 1.000
Post Script complete 0.331
subprocess Interpret
FM Screening complete 0.286
Screening start 0.766
AM Screening complete 0.214
Screening complete 1.000
Current Page Image complete 0.685
Accumulate Images complete 0.860
Rendering complete 0.914
subprocess Render and Screen
Collect Image complete 0.561
Illuminate Document complete 1.000
Move Scan Head complete 0.692
Focus Light Beam complete 0.830
A/D Conversion complete 0.714
Interpolation complete 0.500
Filtered Image complete 0.871
subprocess Capture Image

Figure 6.25: The fuzzy model of the digital copier at the highest level of abstraction. Blue (dark) colored nodes are abstract nodes that can be zoomed in to view the subprocesses underneath. The figure also shows the subprocesses corresponding to the abstract nodes `Interpret`, `Render and Screen`, and `Capture Image`.

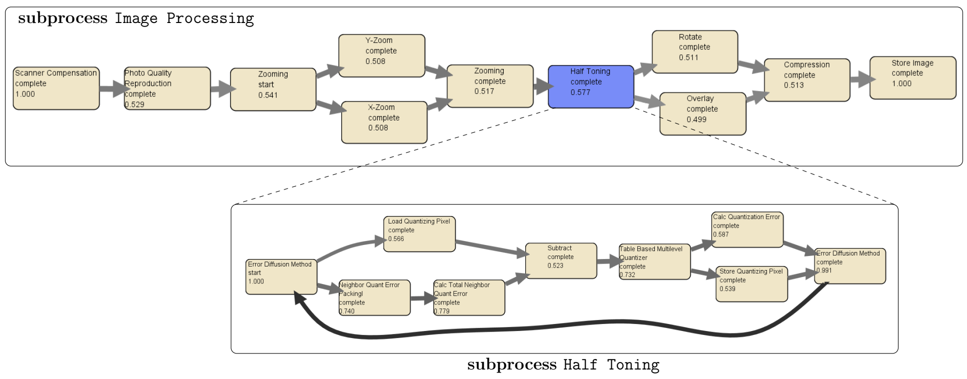

subprocess Image Processing
Scanner Compensation complete 1.000
Photo Quality Reproduction complete 0.529
Zooming start 0.541
Y-Zoom complete 0.508
X-Zoom complete 0.508
Zooming complete 0.517
Half Toning complete 0.577
Rotate complete 0.511
Overlay complete 0.499
Compression complete 0.513
Store Image complete 1.000
Error Diffusion Method start 1.000
Load Quantizing Pixel complete 0.566
Neighbor Quant Error Packing complete 0.740
Calc Total Neighbor Quant Error complete 0.779
Subtract complete 0.523
Table Based Multilevel Quantizer complete 0.732
Calc Quantization Error complete 0.587
Store Quantizing Pixel complete 0.539
Error Diffusion Method complete 0.991
subprocess Half Toning

Figure 6.26: The fuzzy model corresponding to the abstract activity `Image Processing`. This model contains an abstract activity `Half Toning`, the fuzzy model of which is also shown in the figure.

tered along with streets in Amsterdam). In contrast, the abstractions formed using common execution patterns have a strong domain significance. Furthermore, the process map discovered using our two-phase approach is simple (as is evident structurally from the various process models comprising the process map) and has good generalization capabilities, e.g., the ability to scan/print multiple pages is elegantly captured in the loop constructs. We next evaluate the goodness of the proposed approach using the fitness quality dimension in great detail. We evaluate the fitness

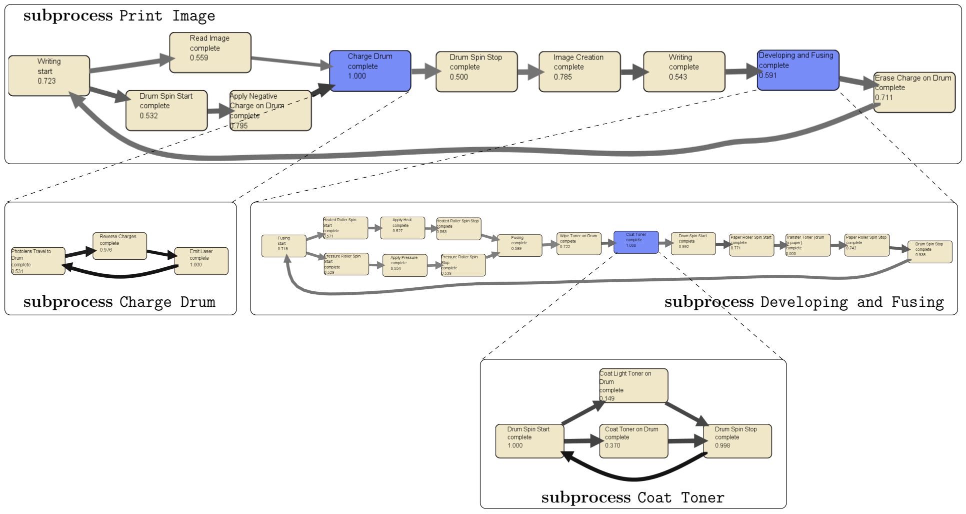
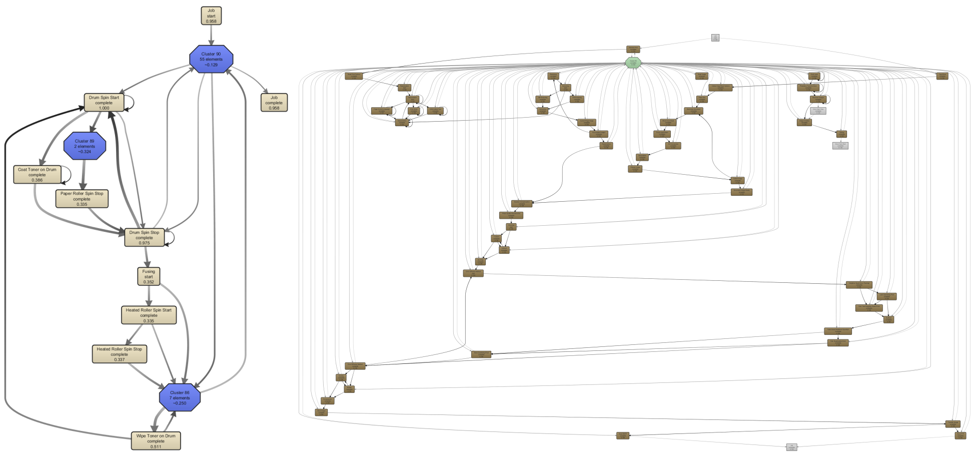

subprocess Print Image
Writing start 0.723
Read Image complete 0.559
Drum Spin Start complete 0.532
Apply Negative Charge on Drum complete 0.795
Charge Drum complete 1.000
Drum Spin Stop complete 0.500
Image Creation complete 0.785
Writing complete 0.543
Developing and Fusing complete 0.591
Erase Charge on Drum complete 0.711
Photolens Travel to Drum complete 0.531
Reverse Charges complete 0.976
Emit Laser complete 1.000
subprocess Charge Drum
subprocess Developing and Fusing
Coat Light Toner on Drum complete 0.149
Drum Spin Start complete 1.000
Coat Toner on Drum complete 0.370
Drum Spin Stop complete 0.998
subprocess Coat Toner

Figure 6.27: The fuzzy model corresponding to the abstract activity `Print Image` and the tree of fuzzy models corresponding to the abstract activities within `Print Image`. This model contains two abstract activities `Charge Drum` and `Developing and Fusing` with the latter containing an abstract activity `Coat Toner` within it.

values of the process map mined using our proposed approach and compare them with the models mined using the traditional approaches.



(a) top-level model (b) process model upon zooming the cluster node 'cluster 90'

Figure 6.28: Process model mined using the classical Fuzzy miner on the raw event log.

Table 6.6 depicts the quality of the mined models using various process discovery algorithms based on the traditional approach, and the two-phase approach (with abstractions). As before, we considered the discovery algorithms that can mine a

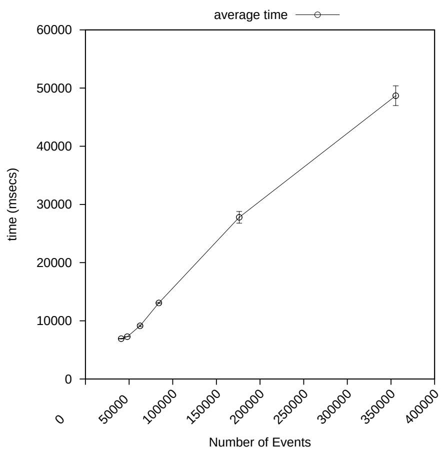
Table 6.6: Comparison between the quality (using fitness as a measure) of the mined models using the traditional approach and the two-phase approach (with abstractions defined over common execution patterns) for the digital copier event log. For the two-phase approach, the fitness values for the top-level model and for each of the subprocesses captured in abstractions are depicted. The α++ miner plug-in in ProM 5.2 ran into an issue (stack overflow due to an infinite loop) when applied on the raw event log (i.e., the log without abstractions). So, no model is discovered and fitness could not be measured (indicated by –). The region miner plug-in in ProM 5.2 takes prohibitively long time to mine the models on this event log and abstraction sub-logs. Hence we did not consider that for our analysis. The fitness values are measured using the conformance checker plug-in in ProM 5.2, which implements the approach presented in [191]. The same discovery algorithm is used across the entire row.

Discovery Algorithm	Traditional Approach	Two-phase Approach										
	Raw Model	Top Level	Inter-pret	Render Screen	Image Proc.	Half Toning	Coat Toner	Charge Drum	Capture Image	Develop. Fusing	Print Image	Avg
α	0.791	0.92	0.94	1.00	1.00	0.88	0.86	1.00	1.00	0.88	0.80	**0.93**
α++	–	1.00	0.96	1.00	1.00	1.00	0.96	1.00	1.00	0.96	0.86	**0.97**
Heuristics Miner	0.89	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.99**
Genetic Miner	0.89	0.92	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.99**
Genetic Miner (with duplicate tasks)	0.68	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	1.00	**0.99**
ILP Miner	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**

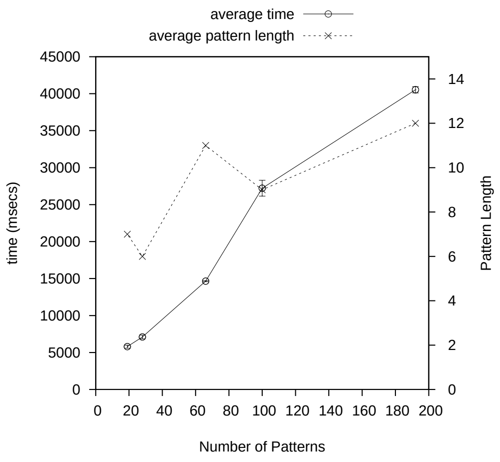
Petri net or a model convertible to a Petri net (heuristic net) from the event log. The 'Raw Model' corresponds to the model mined on the event log without any abstractions. Since the two-phase approach generates multiple models (the top-level model and models for the different abstractions), we consider the average fitness of the different models as a reflection of the overall quality. As shown in Table 6.6, the two-phase approach enables the discovery of models that exhibit a better conformance (fitness) when compared to the traditional approach irrespective of the discovery algorithm adopted. Furthermore, the mined models are less complex and easily comprehensible (scoring on the simplicity factor) and has good generalization capabilities just like in the process map mined using the Fuzzy Map miner (e.g., ability to scan/print multiple pages). The ILP Miner [238] can mine a model that guarantees a fitness of 1.0 for all event logs. Thus we see a fitness of 1.0 for the model mined on the raw event log as well. However, this model is quite spaghetti-like thereby performing poorly on the *simplicity* dimension and is not precise (allows way too much extra behavior).

6.6 Scalability of the Approach

In this section, we study the scalability of the approach proposed for transforming event logs with abstractions. The algorithm for the transformation of logs scans each trace in a log and in the process checks at the current position, the patterns that manifest continuously, non-continuously, and approximately. The total computation time required for the transformation process is influenced both by the size of the event log (the number of traces/events) and the number of patterns. Since we are interested in finding the longest pattern that manifests at a position in a trace, we can optimize the process by first sorting the patterns in the descending order of their length. By doing so, we can stop at the first match (either continuous/non-continuous exact match or approximate match) of any pattern instead of exploring all the patterns at the current position in the trace. We first study the influence of log size on the transformation process by keeping the number of patterns constant. The experiment corresponds to the transformation of the digital copier logs using the abstractions based on the tandem array patterns (the first iteration) defined in Table 6.3. Figure 6.29(a) depicts the average time along with the 95% confidence intervals (over five independent runs) for the transformation of logs of different sizes. We can see that the time for transforming logs varies linearly with respect to the number of events. In order to study the influence of the number of patterns on the transformation process, we keep the size of the log constant. We considered a log containing 100 traces and $40,995$ events pertaining to the digital copier example. We considered the maximal repeat patterns and selected varying number of abstractions. Figure 6.29(b) depicts the average time along with the 95% confidence intervals (over five independent runs) for the transformation of logs for varying number of patterns. The average pattern length is also depicted. We notice that the time for transformation of logs varies linearly with the number of patterns. This indicates that the proposed approach is computationally tractable for large scale event logs.



(a) Influence of log size

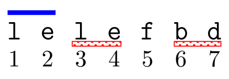


(b) Influence of number of patterns

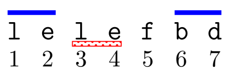
Figure 6.29: Computation time required for the transformation process for varying sizes of the log and the number of patterns.

6.7 Limitations and Extensions

One of the limitations of the proposed approach is the bias introduced in the definition of non-continuous manifestation of patterns (Definition 6.3). The definition prefers the activities in the closest vicinity as the manifestation of a pattern. For example, consider the trace $\underline{t}$ = `lelefbd` and the patterns `le` and `lebd`. Figure 6.30(a) depicts how the pattern manifestations would be considered as per the current definition, i.e., the manifestation of `le` corresponds to $\langle \underline{t}(1), \underline{t}(2) \rangle$ and that of `lebd` corresponds to $\langle \underline{t}(3), \underline{t}(4), \underline{t}(6), \underline{t}(7) \rangle$. One might argue that one could have considered the scenario as depicted in Figure 6.30(b). We propose that such conflicts can be easily resolved by considering additional attributes of the events, e.g., one can consider the data/resource attributes to identify whether the activities `b` and `d` are executed by the same resources as that of the activities `l` and `e` at positions 1 and 2 or 3 and 4; if they happen to be the same as that of 1 and 2, then we can resolve that favorably to the scenario depicted in Figure 6.30(b). Similar argument holds for the disambiguation of abstractions in Algorithm 6.1.



(a)



(b)

Figure 6.30: Resolving conflicts in the non-continuous manifestation of patterns.

6.8 Conclusions

In this chapter, we addressed the problem of dealing with fine-granular event logs and less structured processes. We proposed a two-phase approach to process discovery that enables the discovery of navigable process maps with seamless zoom-in/zoom-out facility. The heart of the approach lies in the first phase where we exploit the abstractions over common execution patterns defined in Chapter 3 and transform the event logs to a desired level of granularity. One can also alternatively use abstractions defined by domain experts. The second phase deals with discovering the process maps with seamless zoom-in/out facility. The definition of abstractions and the transformation of logs can be repetitively applied to mine models with multiple levels of hierarchy. The second phase is a generic phase where any process discovery technique can be applied to mine maps. Using our experiments, we showed that the two-phase approach enables the discovery of high quality models with improved comprehensibility.

Chapter 7
Process Map Performance Analysis

In the previous chapter, we showed how to discover process maps from event logs. Using abstractions, the process can be viewed at different levels of granularity. Process maps play an important role in the analysis of business processes. Process maps can be enriched with additional information such as the performance metrics, which can help us in gaining insights into operational processes, e.g., identifying the bottlenecks in a process, the activities that cause them, etc. Such diagnostics can be used in process redesign or in process optimization. Brand and Van der Kolk [27] have proposed four dimensions, viz., *time*, *cost*, *flexibility*, and *quality*, for analyzing the performance of processes. These dimensions are not necessarily orthogonal, e.g., the cost dimension is closely related to the other dimensions (a low quality process might influence costs through expensive rework). In this chapter, we focus only on the time dimension as it is one of the most fundamental measures of performance [160]. In order to estimate the performance of a process on the time dimension, we need two inputs: (i) a process model/map describing the process and (ii) an event log capturing the process executions, i.e., cases; we assume that the event logs contain information associated with time such as the start/completion timestamp of an event. To analyze the performance based on these inputs, three primary steps need to be taken:

- *define or select appropriate performance metrics and/or Key Performance Indicators (KPIs)*
- *compute the KPIs from event logs via replaying the logs on process maps*
- *project the KPIs onto process maps and highlight potential bottlenecks*

In this chapter, we consider the process maps discovered using the approach presented in Chapter 6 as the input process model. More specifically, we consider Fuzzy maps as the representation for process maps and address all of the above three points. The process maps can be at multiple levels of hierarchy and the input event log can be at any level of hierarchy.

The remainder of this chapter is organized as follows. Related work is presented in Section 7.1. We list interesting *Key Performance Indicators* (KPIs) in Section 7.2. Section 7.3 discusses a technique for replaying event logs onto Fuzzy maps. Relevant KPIs are estimated (measured) during replay. Section 7.4 presents design criteria for annotating Fuzzy maps with the computed KPI metrics. We present and discuss some experimental results in Section 7.5. Finally, Section 7.6 concludes this chapter.

7.1 Related Work

Most of the commercial process monitoring tools compute the performance measures based on either user-defined process models directly linking events in the log to parts of the model or purely from an event log independent of any process model [110]. Currently, very few techniques [2, 110, 225, 240, 251] are available that project performance related information onto *discovered* process models. However, these approaches are restricted to process models that are "flat". In [225], the authors derive performance indicator values from timed workflow logs. The authors first mine a colored workflow (Petri) net from the log using an extension of the α algorithm [229] and subsequently replay the log on the mined net to compute performance measurements. This approach assumes that the discovered model fits the log, i.e., each case in the log should be a trace in the discovered model. Hornix [110] extended the approach presented in [225] by enabling invisible transitions to fire. This facilitates the relaxation of the assumption in [225] so that the model does not have to fit the log. The extension proposed in [110] is based on the log replay approach in [190], although the focus of replay is on measuring conformance between the log and the net rather than on measuring performance. For complex and/or less-structured logs, flat models become "spaghetti-like" showing many details without distinguishing what is important and what is not [218]. Hence, it is difficult for business analysts to obtain/infer any useful insights out of these models.

For classical Fuzzy models [91, 94], animation techniques (based on heuristic replay) are available to visually represent the execution captured in an event log on the model. One can identify potential bottleneck/deadlock activities/flows in the model using this animation. However, this approach does not reveal any explicit performance measures. Moreover, as discussed in the earlier chapter, Fuzzy models support only two-level hierarchies and the user has little control over the abstraction. Moreover, large/less structured processes can still be difficult to comprehend. Van Dongen and Adriansyah [2, 240] have proposed an approach of clustering events to discover a model at a desired level of abstraction and subsequently compute performance measures by replaying a log onto this model. The discovered models are called as Structural Precedence Diagrams (SPDs). Each node in a SPD represents a set of activities performed in a process and there exists many-to-many relationship between nodes and activities. Their replay approach is based on the replay algorithms in [91, 225]. Although SPDs provide a means of dealing with less-structured logs by means of abstractions, it nonetheless is still a flat model. In contrast, the approach proposed in this chapter measures performance indicators and is applicable to process maps with multiple levels of hierarchy. Furthermore, the input event log can be at any level of hierarchy with respect to the process map. We adapt the ideas presented in [2, 240] for the projection of performance information onto process maps. The approach presented in this chapter is an extension of the ideas presented in [197].

7.2 Key Performance Indicators

Key Performance Indicators (KPIs) define a set of values that typically assist in assessing the performance of organizations, processes, departments, resources, etc. These metrics can be derived directly either from an event log or by replaying an event log onto a process map. Several measurements that use timestamps in event logs have been proposed in the literature [2, 110, 116, 182, 212, 225]. Activities in a process can go through different states as depicted in Figure 7.1. An event log with events capturing such fine-grained states of activities enables the estimation of various measures such as the throughput time, service time, synchronization time, and waiting time of cases and activities.

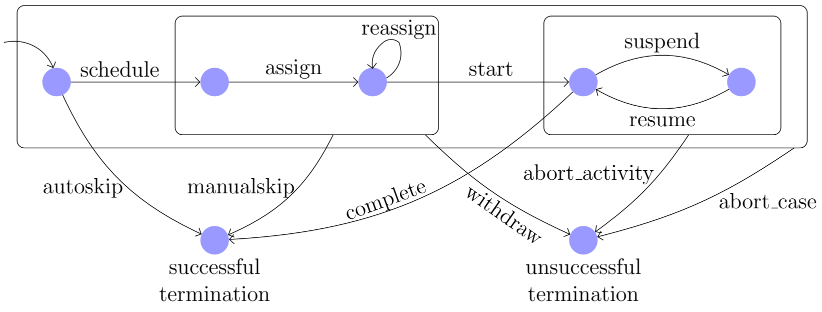


Figure 7.1: Standard transactional life-cycle model [221].

In this chapter, we focus on metrics relevant for annotating process models. We consider the following KPIs, which can be broadly classified into three categories:

1. *Process KPIs:* Process KPIs refer to performance metrics that are measured at the process level or at the level of cases (i.e., process instances) and comprises of the following:
   - *Number of cases:* the total number of cases in the event log $\mathcal{L}$.
   - *Arrival rate of cases:* the number of cases that arrive per unit time.
   - *Case throughput time:* the amount of time it takes to handle a case. This is basically the difference between the time when a case has completed to the time of its creation. Throughput time is also referred to as *lead time*. For a given event log, we can derive measures such as the *minimum*, *maximum*, *average*, and *variance* of case throughput time.
   - *Service levels:* the percentage of cases handled with a predefined time.
2. *Node KPIs:* Nodes in a process map correspond to event classes, e.g., the set of activities. We measure the following metrics for nodes in a process model based on an event log $\mathcal{L}$.

- *Number of executions:* the number of times a node has been invoked or executed among the cases in the event log.
- *Number of skipped executions:* the number of times the execution of a node has been skipped among the cases in the event log.
- *Number of initializations:* a process map can have primitive nodes and abstract nodes. Abstract nodes capture subprocesses. The number of initializations of a node at the top-level of the process map indicate the number of cases in an event log that start at this node. For the nodes inside a subprocess, this corresponds to the number of times instances of the subprocess (within cases) start at the node.
- *Number of terminations:* this is analogous to the number of initializations. For any node in the top-level process model, this metric indicates the number of cases that terminate at the node. If the node is inside a subprocess, this metric indicates the number of times instances of a subprocess terminate at the node.
- *Node throughput time:* the execution of nodes in a process map, which signify event classes, are considered to be atomic. However, abstract nodes in a process map contain a subprocess underneath it. Hence, the execution of abstract nodes is not atomic. The throughput time of abstract nodes corresponds to the total throughput (execution) time of all instances of the subprocess executed in the event log. Apart from the total throughput time, we can also measure the *minimum*, *maximum*, *average*, and *variance* of node throughput times.

3. *Edge KPIs:* We measure the following KPIs for the edges in a process map
    - *Number of executions:* for any edge in the process map, this corresponds to the number of times the edge is executed in an event log, i.e., control is passed from the source node to the target node of the edge.
    - *Number of skipped executions:* for any edge in the process map, this corresponds to the number of times execution of the edge is skipped. This arises when either the source node or the target node of the edge is skipped in an event log.
    - *Edge throughput time:* for any edge in the process map, this corresponds to the total time spent to route control among all instances of execution of the edge in an event log. For each instance of execution of the edge, this corresponds to the difference between the timestamps of the event instance associated to the target node and the event instance associated to the source node. Apart from the total throughput time, we can also measure the *minimum*, *maximum*, *average*, and *variance* of edge throughput times.

In the next section, we present an approach to measure these KPIs by replaying an event log on a process map.

7.3 Replaying Logs on Process Maps

As discussed in the previous chapter, a process map can have abstract nodes. Abstract nodes can be zoomed-in to view the subprocesses captured by them. Subprocesses captured by an abstract node can in turn have abstract nodes inside them thereby inducing a hierarchy of processes. A process map with hierarchies can be visualized as a tree as depicted in Figure 7.2. Each node in the tree represents a process model in the process map and the depth of a node indicates the level of the corresponding process model in the process map. The tree depicted in Figure 7.2 represents a process map with four hierarchical levels. The root node M_0 indicates the top-level process model and is considered to be at level 0. The top-level process model contains $n+1$ subprocesses $M_{10}, M_{11}, \ldots, M_{1n}$, which are at level 1 of the hierarchy. The subprocesses M_{11} and M_{1n} have further abstract nodes within it, which are captured by the process models M_{20}, M_{21}, and M_{23} at level 2 of the hierarchy. Finally, the process model M_{21} has an abstract node defining the subprocess M_{30}, which is at level 3 of the hierarchy.

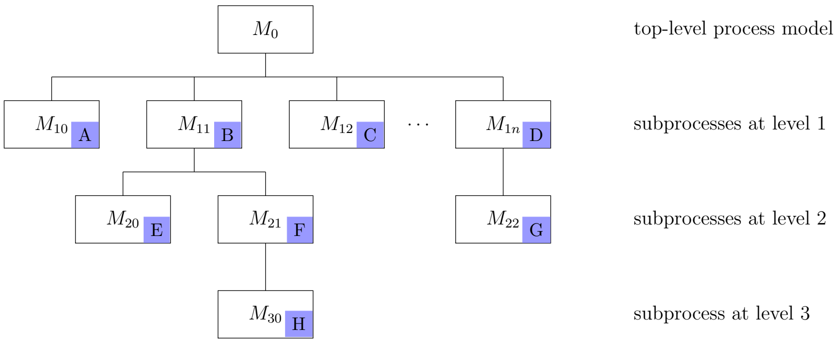


Figure 7.2: A process map visualized as a tree of process models. Subprocesses are depicted with their corresponding abstract activity manifested as a node in its parent model. For example, the subprocess M_{30} corresponds to the abstract activity `H`, which is a node in M_{21}.

We choose Fuzzy maps as the representation for process maps. *We propose an approach using principles from graph theory for replaying an event log on the Fuzzy maps. We assume that the process map can have any number of hierarchical levels and that the event log can be at any level of abstraction with respect to the process map.* Furthermore, *we assume that each process instance can be replayed on the process map to a large extent*, i.e., we assume that the noise (if any) in the event logs is negligible. Before we look at the replay technique, we define a few terms.

A process map defines a set of process models (as depicted in Figure 7.2). Let $\mathbb{M}$ denote the set of all process models in a process map. Let $\mathcal{AM}$ denote the set of all activities in the process map (each node in a process map corresponds to an activity). Let $l_m : \mathbb{M} \to \mathbb{N}_0$ denote the level of a process model. Let $f : \mathbb{M} \to 2^{\mathcal{AM}}$ denote the set of activities in a process model $M \in \mathbb{M}$. *It could be the case that there are duplicate activities in the process map, i.e., two or more models can have some* *activities in common.* A model can be unambiguously characterized by the activities that belong exclusively to that model.

Definition 7.1 (Activities Unique to a Model). Given a process map defined by a set of process models $\mathbb{M}$ and the set of activities $\mathcal{AM}$, the *activities unique to a process model* $M \in \mathbb{M}$, $u : \mathbb{M} \to 2^{\mathcal{AM}}$, is defined as the set of activities that are present only in this process model M. In other words,

$$u(M) = \{a \in f(M) \mid \forall_{\substack{M' \in \mathbb{M} \\ M' \neq M}} a \notin f(M')\}$$

⌟

Definition 7.2 (Level of an Event Log with Respect to a Process Map). Given a process map defined by a set of process models $\mathbb{M}$ and the set of activities $\mathcal{AM}$, and an event log $\mathcal{L} \in \mathcal{B}(\mathcal{A}^*)$[1], the level of the log $\mathcal{L}$ with respect to the process map, is defined as the deepest level of a model $M \in \mathbb{M}$ such that $\mathcal{L}$ contains at least one activity from the set of activities unique to the model M. In other words,

$$\text{Level of a log } \mathcal{L} = \max\{l \mid l \in l_m(M) \wedge \mathcal{A} \cap u(M) \neq \varnothing \text{ for all } M \in \mathbb{M}\}$$

⌟

We now discuss our replay technique. The required inputs are a Fuzzy map and an event log. The event log can be at any level of hierarchy with respect to the process map. *The basic idea of our replay technique is to process the event log iteratively from the deepest level to the top-most level (on lines similar to the transformation of logs defined in Section 6.3). During this process we relate manifestations of activity sequences to flows in the process models at the corresponding level and compute the relevant KPIs. At the same time, the event log is transformed into a higher-level of abstraction by replacing the mapped activity sequence with its abstract activity, which captures the process model to which the activity sequence is mapped to. In addition, we set the start and completion timestamps of the abstract activity to the timestamps of the first and the last event in the activity sequence respectively. The transformed event log forms the input for the next iteration.* It is important to note that at any iteration, only the activities in the event log belonging to that level are processed and activities at higher levels are postponed for processing at appropriate levels. Figure 7.3 depicts the basic idea of our replay technique.

There are two core aspects of our replay technique:

1. identifying the manifestation of activity sequences
2. mapping the activity sequences to flows in a Fuzzy model

For the first step, we exploit the common execution patterns (maximal repeats and tandem arrays defined in Chapter 3) to guide the process of replay. This is unlike the approach presented in [2, 240] where the authors use a sliding window of fixed size over a trace in identifying the activity manifestations. For the second step, we take *guided walks* through the graph (Fuzzy model) and identify the flows; the walks

[1]here, we assume that all elements of $\mathcal{A}$ appear in $\mathcal{L}$.

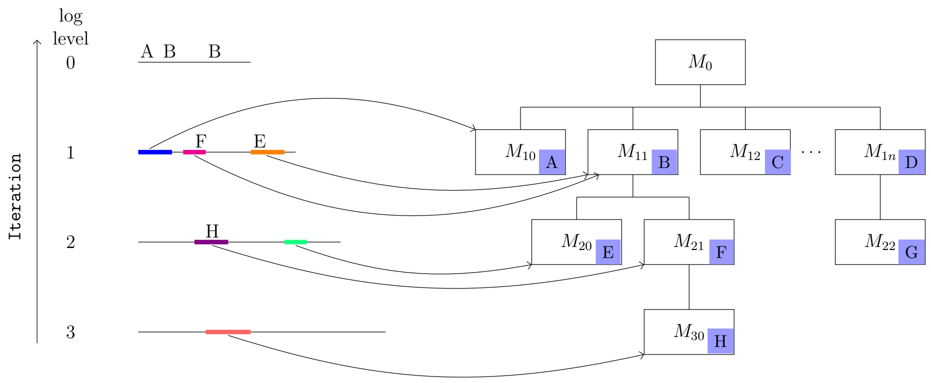


Figure 7.3: The basic idea of replaying a log onto a process map. Activities in an event log are processed from the lowest level of abstraction to the highest level. At any level, the common execution patterns manifested in an event log are mapped onto the flows in an appropriate model at that level and performance metrics are computed.

are guided by the activity sequence.

Algorithms 7.1 and 7.2 present a high-level overview of our approach to replay an event log onto a Fuzzy map. We discuss the algorithm using the same example as used to illustrate the discovery of process maps in Section 6.4. Figure 7.4 depicts the Fuzzy map and Figure 7.5 depicts the tree of Fuzzy models in the map. The Fuzzy map consists of three hierarchical levels. Let us consider an event log at the deepest level $\mathcal{L} = [\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3, \mathtt{t}_4, \ldots]$ whose corresponding sequences of activities are $[\mathtt{jgclebdabdfi}, \mathtt{jgcfahbdledi}, \mathtt{gjahbdfi}, \mathtt{cgjlefbdi}, \ldots]$ defined over the set of activities $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{l}\}$ (Step 1, Algorithm 7.1). As depicted in Figure 7.5, the Fuzzy map contains the Fuzzy models $\mathbb{M} = \{M_0, M_{10}, M_{11}, M_{20}, M_{21}\}$ and contains the activities $\mathcal{AM} = \{\mathtt{Artificial\ Start\ Task}, \mathtt{Artificial\ End\ Task}\} \cup \mathcal{A}$ (Step 2, Algorithm 7.1). $f(M)$ defines the set of activities in a process model $M \in \mathbb{M}$, e.g., $f(M_0) = \{\mathtt{Artificial\ Start\ Task}, \mathtt{Artificial\ End\ Task}, \mathtt{W}, \mathtt{M}, \mathtt{f}, \mathtt{i}\}$ (Step 3, Algorithm 7.1). The level of the log $\mathcal{L}$, $level_{\mathcal{L}}$, with respect to the process map is 2 (Step 4, Algorithm 7.1).

The algorithm iterates over the log from the deepest level to the top level and considers only the process models at the current log level in any iteration (Step 5, Algorithm 7.1). M_{20}, and M_{21} are the process models at level 2 and the set of activities in these two models are $\mathcal{AM}_2 = \{\mathtt{Artificial\ Start\ Task}, \mathtt{Artificial\ End\ Task}, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{h}, \mathtt{l}\}$ (Step 6, Algorithm 7.1). Step 7 of the algorithm discovers the set of common execution patterns in the event log. As mentioned earlier, we consider the tandem arrays and maximal repeats as the notion of common execution patterns. $P = \{\mathtt{jgc}, \mathtt{lebd}, \mathtt{le}, \mathtt{ahbd}, \mathtt{bd}, \mathtt{di}, \mathtt{f}, \mathtt{bdfi}, \ldots\}$ are some of the maximal repeats in $\mathcal{L}$. We prune the patterns P to reflect only the patterns involving the activities at this level. Pruning here refers to ignoring patterns that do not involve any activity from this level and the removing of activities not belonging to this level from a

Algorithm 7.1 Replay Algorithm to Compute Performance

```
1: Let $\mathcal{L} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n]$ be an event log defined over the set of events $\mathcal{E}$ and the set of activities $\mathcal{A}$.
2: Let $\mathcal{M}$ be a process map, $\mathbb{M}$ be the set of process models in the process map, and $\mathcal{AM}$ be the set of activities in the process map. $M_0 \in \mathbb{M}$ is the top-level process model.
3: Let $f : \mathbb{M} \rightarrow 2^{\mathcal{AM}}$ define the set of activities belonging to a process model $M \in \mathbb{M}$.
4: Let $level_{\mathcal{L}}$ denote the level of the log with respect to the process map $\mathcal{M}$
5: for $l = level_{\mathcal{L}}$ to 1 do
6:     Let $\mathbb{M}_l \subseteq \mathbb{M}$ be the set of process models at level $l$ and $\mathcal{AM}_l \subseteq \mathcal{AM}$ be the set of activities in $\mathbb{M}_l$, i.e,. $\mathcal{AM}_l = \bigcup_{M \in \mathbb{M}_l} f(M)$
7:     Let $P$ be the set of common execution patterns in $\mathcal{L}$. Prune $P$ to reflect only patterns involving the activities in the process models at this level. Let $P_l$ be the set of such pruned patterns.
8:     Initialize $\mathcal{L}' = [\ ]$
9:     for all $\mathbf{t} \in \mathcal{L}$ do
10:        Let $\underline{\mathbf{t}}$ correspond to the sequence of activities in $\mathbf{t}$
11:        Let $\mathbf{t}' = \langle\rangle$ be the transformed trace corresponding to $\mathbf{t}$
12:        for $i = 1$ to $|\underline{\mathbf{t}}|$ do
13:            if $\underline{\mathbf{t}}(i) \notin \mathcal{AM}_l$ then
14:                append $\mathbf{t}(i)$ to the transformed trace, i.e., $\mathbf{t}' = \mathbf{t}' \diamond \mathbf{t}(i)$
15:            end if
16:
17:            Let $\mathbf{p} \in P$ be the longest manifestation of a pattern (considering the continuous, non-continuous, and approximate manifestations) at position $i$ in $\underline{\mathbf{t}}$
18:            If $\mathbf{p}$ is manifested either non-continuously or approximately at position $i$, then readjust its manifestation in both $\mathbf{t}$ and $\underline{\mathbf{t}}$
19:            Let $M$ be the model that can be best associated to pattern $\mathbf{p}$. Prune $\mathbf{p}$ to contain only activities belonging to $M$
20:
21:            Map_and_Update_Performance($\mathbf{p}$, $M$, $\mathbf{t}$, $i$, $\mathbf{t}'$)
22:
23:            Set $i = i + |\mathbf{p}| - 1$
24:        end for
25:        $\mathcal{L}' = \mathcal{L}' \uplus [\mathbf{t}']$
26:    end for
27:    Set $\mathcal{L} = \mathcal{L}'$
28: end for
29:
30: Find patterns in the log at level 0, map and update performance metrics at the top-level process model
```

Algorithm 7.2 Map_and_Update_Performance

```
Require: a pattern $\mathbf{p}$, a model $M$, trace $\mathbf{t}$, pattern index $i$ in trace $\mathbf{t}$, transformed trace $\mathbf{t}'$
 1: get all guided walks defined by the pattern $\mathbf{p}$ in the model $M$. For subsequences in $\mathbf{p}$ for which a walk does not exist, get all the individual activities
 2: for each walk $\mathbf{w}$ defined by the pattern do
 3:     update the node and edge metrics
 4: end for
 5: for each activity in $\mathbf{p}$ not mapped to any walk do
 6:     update the node metrics
 7: end for
 8: if there exists multiple walks and/or activities not mapped to any walk then
 9:     determine the connectivity between the different walks and individual activities not mapped to any walk defined by the pattern $\mathbf{p}$ in $M$ and update the node and edge metrics pertaining to the connectivity
10: end if
11: if a new instance of $M$ has to be created then
12:     create a new event $e$ with the activity name as the abstraction corresponding to $M$; set the start timestamp of the event $e$ to that of the first event in $\mathbf{p}$ and the complete timestamp to that of the last event in $\mathbf{p}$;
13:     append the event $e$ to the transformed trace $\mathbf{t}'$, i.e., $\mathbf{t}' = \mathbf{t}' \diamond e$
14: else
15:     determine the connectivity between the previous instance of $M$ and the pattern $\mathbf{p}$ and update the node and edge metrics pertaining to the connectivity
16:     if $M$ is a subprocess then
17:         update the completion timestamp of the event $e$ corresponding to the last instance of $M$ in $\mathbf{t}'$
18:     end if
19: end if
```

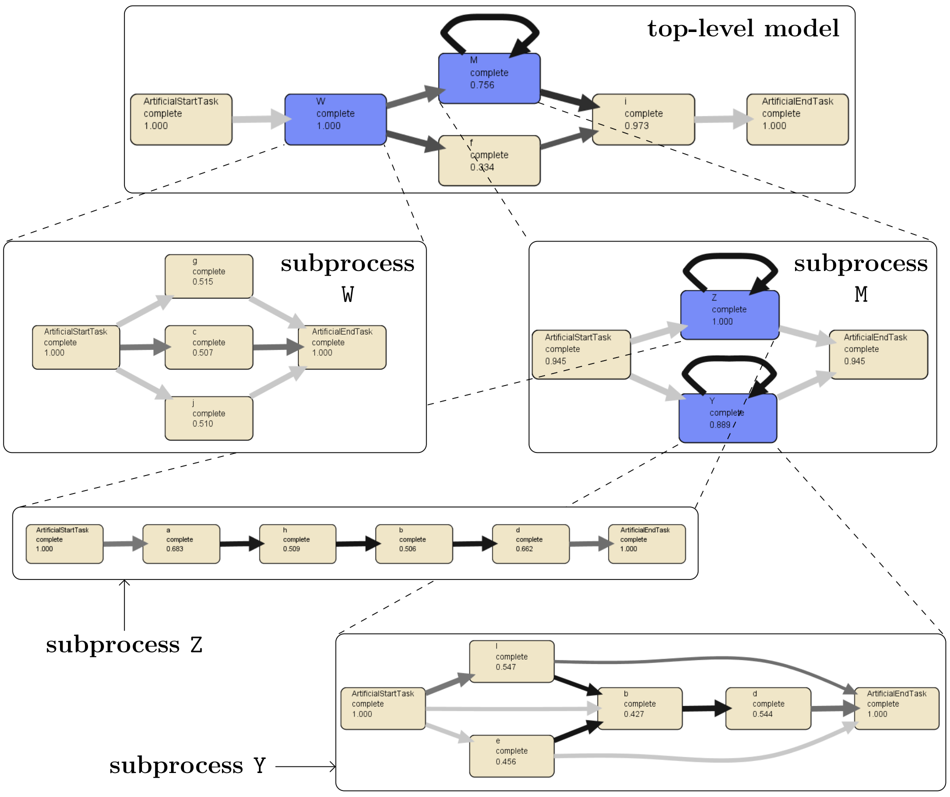


Figure 7.4: Process map of a simple example discussed in Section 6.4.

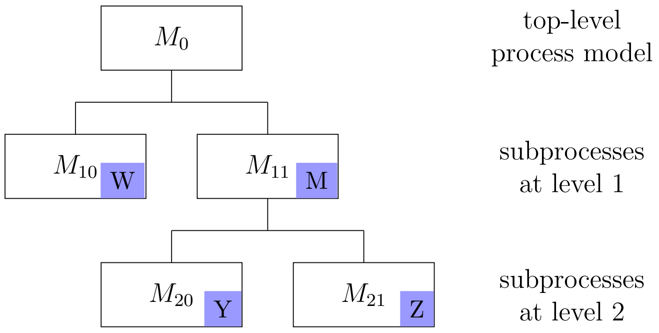
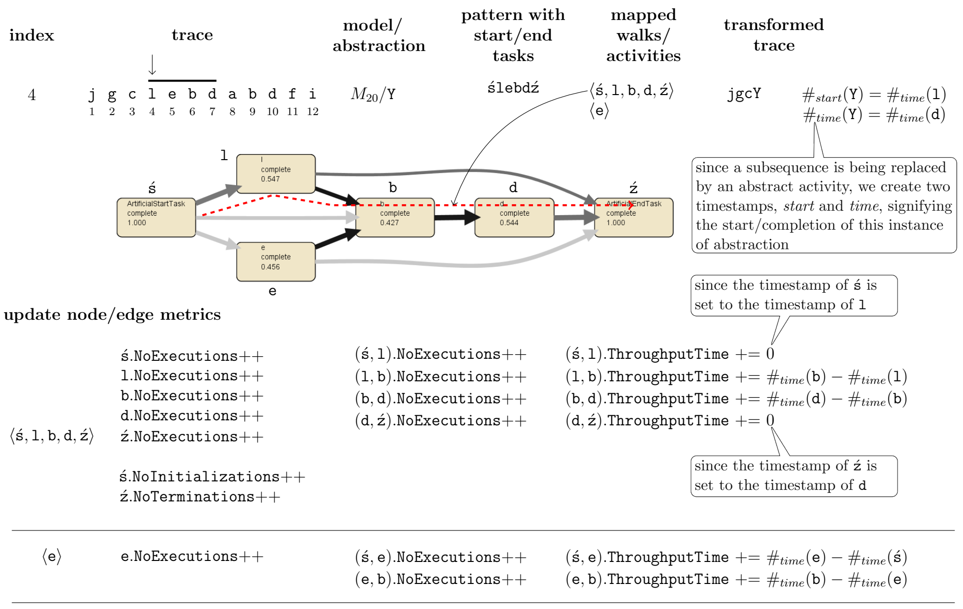


Figure 7.5: Process map depicted in Figure 7.4 visualized as a tree of process models. Subprocesses are annotated with their corresponding abstract activity.

pattern $\mathbf{p} \in P$ that contains some activities from this level $\mathcal{AM}_2$. For example, the pattern `jgc` $\in P$ is ignored because it is defined over activities not in $\mathcal{AM}_2$, the pattern `bdfi` is pruned to `bd` by removing `f` and `i` that do not belong to $\mathcal{AM}_2$. The pruned patterns $P_2 = \{$`ledb`, `le`, `bd`, `ahbd`, `d`, `bd`$\}$. The steps so far comprise the initial-



index	trace	model/ abstraction	pattern with start/end tasks	mapped walks/ activities	transformed trace
4	j g c l e b d a b d f i (1 2 3 4 5 6 7 8 9 10 11 12)	M_{20}/Y	ślebdź	⟨ś, l, b, d, ź⟩ ⟨e⟩	jgcY $\#_{start}(\texttt{Y}) = \#_{time}(\texttt{l})$ $\#_{time}(\texttt{Y}) = \#_{time}(\texttt{d})$

since a subsequence is being replaced by an abstract activity, we create two timestamps, *start* and *time*, signifying the start/completion of this instance of abstraction

update node/edge metrics

since the timestamp of ś is set to the timestamp of l

⟨ś, l, b, d, ź⟩	ś.NoExecutions++	(ś, l).NoExecutions++	(ś, l).ThroughputTime += 0
	l.NoExecutions++	(l, b).NoExecutions++	(l, b).ThroughputTime += $\#_{time}(\texttt{b}) - \#_{time}(\texttt{l})$
	b.NoExecutions++	(b, d).NoExecutions++	(b, d).ThroughputTime += $\#_{time}(\texttt{d}) - \#_{time}(\texttt{b})$
	d.NoExecutions++	(d, ź).NoExecutions++	(d, ź).ThroughputTime += 0
	ź.NoExecutions++		
	ś.NoInitializations++		
	ź.NoTerminations++		
⟨e⟩	e.NoExecutions++	(ś, e).NoExecutions++	(ś, e).ThroughputTime += $\#_{time}(\texttt{e}) - \#_{time}(\texttt{ś})$
		(e, b).NoExecutions++	(e, b).ThroughputTime += $\#_{time}(\texttt{b}) - \#_{time}(\texttt{e})$

since the timestamp of ź is set to the timestamp of d

Figure 7.6: Mapping of the pattern `lebd` at position 4 in the trace $\underline{\mathbf{t}_1}$ = `jgclebdabdfi` onto the process model M_{20} and updating the node and edge metrics.

ization phases of the algorithm and the actual processing of traces starts from Step 9.

The basic idea here is to scan each trace from left to right and process the manifestation of patterns at this level in the trace while postponing the processing of activities not in this level for later stages. In the process, the pattern manifestation is mapped onto an appropriate model at this level and the performance metrics are updated. During this process traces in the event log are transformed into higher-levels of abstraction. The transformed trace captures all postponed activities and the abstract activity corresponding to each pattern manifestation in the original trace that is mapped to a model. In the next iteration, the event log corresponds to the bag of transformed traces.

Let us consider the trace $\mathbf{t}_1$ with the corresponding activity sequence $\underline{\mathbf{t}_1}$ = `jgcleb-dabdfi` (Step 10, Algorithm 7.1). Let $\mathbf{t}'_1$ denote the transformed trace corresponding to $\mathbf{t}$ and $\underline{\mathbf{t}'_1}$ be its activity sequence (Step 11, Algorithm 7.1). Since the first three activities in $\underline{\mathbf{t}}$ do not belong to the current log level, they are appended to the transformed trace, i.e., $\underline{\mathbf{t}'_1}$ = `jgc` (Steps 13–15, Algorithm 7.1). For simplicity reasons, we depict only the activity sequences to explain the concept. At position 4 (see Figure 7.6), we have a continuous manifestation of the pattern $\mathbf{p}$ = `lebd` (Step 17, Algorithm 7.1). The process model M_{20}, whose corresponding abstract activity is `Y`, contains all of the activities involved in the pattern and hence $\mathbf{p}$ is best associated to M_{20} (Step 19, Algorithm 7.1). We next need to map the pattern $\mathbf{p}$ to the flows in the model M_{20} (Step 21, Algorithm 7.1).

Since the models have artificial start/end nodes whereas the event log and patterns do not, we append the artificial start/end activities to the pattern before creating guided walks on the model. The timestamp of the artificial start task is set to the timestamp of the first activity in $\mathbf{p}$ and that of the artificial end task is set to that of the last activity in $\mathbf{p}$. For ease of use, we represent the artificial start and end tasks by $\texttt{ś}$ and $\texttt{ź}$. The pattern after appending the artificial start/end tasks becomes $\texttt{ślebdź}$. We now do a walk over the graph $G = (V, E)$ (corresponding to M_{20}) where the walk is guided by the pattern. In other words, *we scan the pattern from left to right and walk along the nodes and edges in the graph corresponding to the sequence defined by the pattern. If at any node $v_p \in V$ in the graph, we do not find an edge/path leading to the next activity in the pattern, we skip the activities in the pattern until an activity corresponding to node v_q is found such that there is an edge/path from v_p to v_q. In other words, $(v_p, v_q) \in E$ or there exists a path $\langle v_p, v_{p+1}, \ldots, v_q \rangle$ in G. If some activities have been skipped in the pattern, we try to find walks over the sequence comprising the skipped activities. We repeat this procedure until the entire pattern is mapped or until we are left with a sequence for which no walk exists.* We then update the edge/node metrics for each of the walks over the pattern and also update the edge/node metrics connecting the different walks.

For the given pattern $\texttt{ślebdź}$, there exists a walk $\langle \texttt{ś}, \texttt{l}, \texttt{b}, \texttt{d}, \texttt{ź} \rangle$ covering the pattern in the graph corresponding to the process model M_{20} (Step 1, Algorithm 7.2). At node $\texttt{l}$ the next activity in the pattern is $\texttt{e}$. However, we do not see an edge/path from $\texttt{l}$ to $\texttt{e}$ in the graph. So, we skip $\texttt{e}$ from the pattern and proceed further, i.e., the activity $\texttt{b}$ next to $\texttt{e}$ for which we have an edge from $\texttt{l}$. Proceeding further, we find a walk that covers the rest of the pattern. We now visit the skipped activities in the pattern during the walk. In this case, we are left with only one skipped activity $\texttt{e}$. Since we cannot find a walk involving only one activity, we terminate the process with the walk $\langle \texttt{ś}, \texttt{l}, \texttt{b}, \texttt{d}, \texttt{ź} \rangle$ and an individual activity $\langle \texttt{e} \rangle$. We update the edge/node metrics corresponding to the walk (flow) in the graph (Steps 2–4, Algorithm 7.2). For the node metrics, we increment the number of executions of all nodes defined in the walk by 1, so, the number of executions of $\texttt{ś}$, $\texttt{l}$, $\texttt{b}$, $\texttt{d}$, and $\texttt{ź}$ is incremented by 1. If a node is an abstract node, we increment its throughput time as well. Since this walk does not contain abstract nodes, this does not apply here. We will look at this in later stages (at subsequent log levels) for this example.

For each edge defined by the walk, we update both the number of executions and the edge throughput time. In other words, the number of executions of the edges $(\texttt{ś}, \texttt{l})$, $(\texttt{l}, \texttt{b})$, $(\texttt{b}, \texttt{d})$, and $(\texttt{d}, \texttt{ź})$ is incremented by 1. The edge throughput time of $(\texttt{ś}, \texttt{l})$ is incremented by 0 since the time of the artificial start task, $\texttt{ś}$, is set to the timestamp of the first activity $\texttt{l}$ (i.e., $\#_{time}(\mathtt{t}_1(4))$). The throughput times of the edges $(\texttt{l}, \texttt{b})$ and $(\texttt{b}, \texttt{d})$ are incremented by $\#_{time}(\mathtt{t}_1(6)) - \#_{time}(\mathtt{t}_1(4))$ and $\#_{time}(\mathtt{t}_1(7)) - \#_{time}(\mathtt{t}_1(6))$ respectively. The throughput time of the edge $(\texttt{d}, \texttt{ź})$ is incremented by 0 since the timestamp of the artificial end task, $\texttt{ź}$, is set to the timestamp of $\texttt{d}$ (i.e., $\#_{time}(\mathtt{t}_1(7))$). We now process the activity $\texttt{e}$ not mapped to any walk in the pattern (Steps 5–7, Algorithm 7.2). From the graph, we identify the nodes in the graph that connect the activity to an already mapped walk. The edges $(\acute{\mathtt{s}}, \mathtt{e})$ and $(\mathtt{e}, \mathtt{b})$ connect the skipped activity e to the walk (Steps 8–10, Algorithm 7.2). We increment the number of executions of the node e and the number of executions of the edges $(\acute{\mathtt{s}}, \mathtt{e})$ and $(\mathtt{e}, \mathtt{b})$ by 1. We also increment the throughput times of the edges by $\#_{time}(\mathbf{t}_1(5)) - \#_{time}(\mathbf{t}_1(4))$ and $\#_{time}(\mathbf{t}_1(6)) - \#_{time}(\mathbf{t}_1(5))$ respectively.

Since this pattern corresponds to the abstract activity Y, we create an event e with activity name as Y (Steps 11–12, Algorithm 7.2). The timestamp attribute of this event is set to the timestamp of the last event in the pattern, i.e., $\#_{time}(e) = \#_{time}(\mathbf{t}_1(7))$. Since an abstract activity captures a subsequence of events, we create an additional attribute that signifies the start timestamp of the event sequence and set its value to the first event's timestamp in the pattern, i.e., $\#_{start}(e) = \#_{time}(\mathbf{t}_1(4))$. This is essential when updating the throughput times of the abstract nodes and the throughput times of the edges leading to/from the abstract node. We add the event e to the transformed traced $\mathbf{t}'_1$, i.e., $\underline{\mathbf{t}'_1} = \mathtt{jgcY}$ (Step 13, Algorithm 7.2). i is incremented to 7 (Step 23, Algorithm 7.1).

At position 8 (see Figure 7.7), we have an approximate manifestation of the pattern ahbd as $\mathbf{p} = \mathtt{abd}$ (Step 17, Algorithm 7.1). This pattern is mapped to M_{21}. Just like in the above case, we add an artificial start/end task to the pattern. The pattern after this addition becomes $\acute{\mathtt{s}}\mathtt{abd}\acute{\mathtt{z}}$. We now do a walk over the graph $G = (V, E)$ (corresponding to M_{21}) where the walk is guided by the pattern. There exists a walk $\langle \acute{\mathtt{s}}, \mathtt{a}, \mathtt{h}, \mathtt{b}, \mathtt{d}, \acute{\mathtt{z}} \rangle$ capturing the pattern (Step 1, Algorithm 7.2). Note that at node a the next activity in the pattern is b but there is no edge that directly connects a and b. However, we have a path $\langle \mathtt{a}, \mathtt{h}, \mathtt{b} \rangle$ connecting a and b. Now we update the node and edge metrics (Steps 2–4, Algorithm 7.2). The number of executions of the nodes $\acute{\mathtt{s}}, \mathtt{a}, \mathtt{b}, \mathtt{d}$, and $\acute{\mathtt{z}}$ is incremented by 1. Since we have an activity h in the walk that is not in the pattern, the node h is considered to be *skipped*. We increment the skip frequency of h by 1. The number of executions of the edges $(\acute{\mathtt{s}}, \mathtt{a})$, $(\mathtt{b}, \mathtt{d})$, and $(\mathtt{d}, \acute{\mathtt{z}})$ is incremented by 1 while the edges $(\mathtt{a}, \mathtt{h})$ and $(\mathtt{h}, \mathtt{b})$ have their skip frequency incremented by 1. The throughput time of the edges $(\acute{\mathtt{s}}, \mathtt{a})$, $(\mathtt{b}, \mathtt{d})$, and $(\mathtt{d}, \acute{\mathtt{z}})$ is incremented by 0, $\#_{time}(\mathbf{t}_1(10)) - \#_{time}(\mathbf{t}_1(9))$, and 0 respectively (recall that the timestamp of $\acute{\mathtt{s}}$ is set to that of a and the timestamp of $\acute{\mathtt{z}}$ is set to that of d). We create an event e whose activity name corresponds to Z (Steps 11–12, Algorithm 7.2). The timestamp attribute of this event is set to the timestamp of the last event in the pattern, i.e., $\#_{time}(e) = \#_{time}(\mathbf{t}_1(10))$ and the start time of this event is set to $\#_{start}(e) = \#_{time}(\mathbf{t}_1(8))$. The event e is appended to the transformed trace $\mathbf{t}'_1$, i.e., $\underline{\mathbf{t}'_1} = \mathtt{jgcYZ}$ (Step 13, Algorithm 7.2). i is updated to 10 (Step 23, Algorithm 7.1).

In the next iterations, the activities f and i at positions 11 and 12 do not belong to this level. Therefore, we append them to the transformed trace (Steps 13–15, Algorithm 7.1). The transformed trace after the end of processing all activities at this level is $\underline{\mathbf{t}'_1} = \mathtt{jgcYZfi}$, which is at level 1. Proceeding further, we process all the traces in the event log.

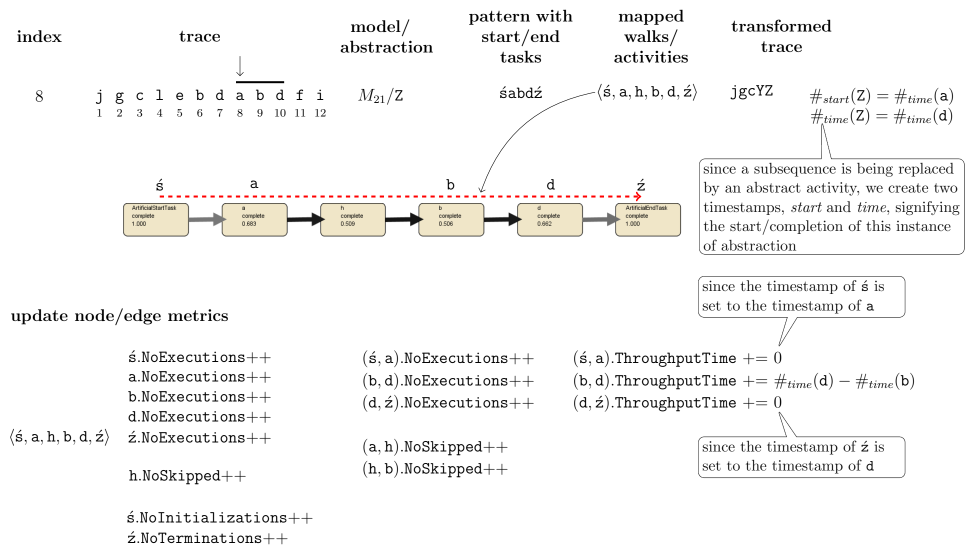

index
trace
model/ abstraction
pattern with start/end tasks
mapped walks/ activities
transformed trace
8
j g c l e b d a b d f i
1 2 3 4 5 6 7 8 9 10 11 12
M_{21}/Z
śabdź
⟨ś, a, h, b, d, ź⟩
jgcYZ
$\#_{start}$(Z) = $\#_{time}$(a)
$\#_{time}$(Z) = $\#_{time}$(d)
since a subsequence is being replaced by an abstract activity, we create two timestamps, *start* and *time*, signifying the start/completion of this instance of abstraction
ś
a
b
d
ź
ArtificialStartTask complete 1.000
a complete 0.683
h complete 0.509
b complete 0.506
d complete 0.662
ArtificialEndTask complete 1.000
since the timestamp of ś is set to the timestamp of a
update node/edge metrics
⟨ś, a, h, b, d, ź⟩
ś.NoExecutions++
a.NoExecutions++
b.NoExecutions++
d.NoExecutions++
ź.NoExecutions++
h.NoSkipped++
ś.NoInitializations++
ź.NoTerminations++
(ś, a).NoExecutions++
(b, d).NoExecutions++
(d, ź).NoExecutions++
(a, h).NoSkipped++
(h, b).NoSkipped++
(ś, a).ThroughputTime += 0
(b, d).ThroughputTime += $\#_{time}$(d) − $\#_{time}$(b)
(d, ź).ThroughputTime += 0
since the timestamp of ź is set to the timestamp of d

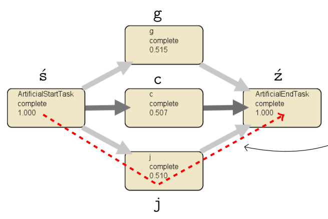
Figure 7.7: The pattern **ahbd** has an approximate manifestation, **abd**, at position 8 in the trace $\underline{\texttt{t}_1}$ = **jgclebdabdfi**. Mapping the manifestation **abd** onto the process model M_{21} and updating the node and edge metrics.

Now we process the traces at level 1 (Step 5, Algorithm 7.1). There are two process models M_{10} and M_{11} corresponding to abstractions W and M respectively at level 1. Some of the common execution patterns in the event log at this level are P_1 = {jgc, gj, YZ, Y, Z, . . . }. Consider the trace $\underline{\texttt{t}_1}$ = jgcYZfi at this level. At position 1 (see Figure 7.8), there exists a continuous manifestation of the pattern **p** = jgc in $\underline{\texttt{t}_1}$ (Step 17, Algorithm 7.1). The process model M_{10} corresponding to the abstract activity W contains all of the activities involved in the pattern and hence **p** is best associated to M_{10} (Step 19, Algorithm 7.1). We next need to map the pattern **p** to the flows in the model M_{10} (Step 21, Algorithm 7.1). Since the model contains an artificial start/end task, we augment the pattern with these activities. The pattern after adding the artificial start/end activities becomes **p** = śjgcź. We now do a guided walk over the graph $G = (V, E)$ (corresponding to M_{10}). For the given pattern śjgcź, there exists a walk ⟨ś, j, ź⟩ covering the pattern (Step 1, Algorithm 7.2). At node j the next activity in the pattern is g. However, we do not see an edge/path from j to g in the graph. So, we skip g from the pattern and proceed further. We do not see an edge/path from j to c either. We skip c as well. We find an edge from j to ź thereby terminating the walk ⟨ś, j, ź⟩. Now, we revisit the skipped activities. The sequence corresponding to the skipped activities is gc. We do not find a walk corresponding to gc. Therefore, the process terminates with a walk ⟨ś, j, ź⟩ and two individual activities ⟨g⟩ and ⟨c⟩.

We update the edge/node metrics corresponding to the walk (flow) (Steps 2–4, Algorithm 7.2) and the activities in the pattern not mapped to any walk (Steps 5–7, Algorithm 7.2). In other words, we increment the number of executions of

index	trace	model/ abstraction	pattern with start/end tasks	mapped walks/ activities	transformed trace	
1	$\downarrow$ $\underline{\texttt{j g c Y Z f i}}$ (1 2 3 4 5 6 7)	M_{10}/W	śjgcź	⟨ś, j, ź⟩ ⟨g⟩ ⟨c⟩	W	$\#_{start}(\texttt{W}) = \#_{time}(\texttt{j})$ $\#_{time}(\texttt{W}) = \#_{time}(\texttt{c})$

since a subsequence is being replaced by an abstract activity, we create two timestamps, *start* and *time*, signifying the start/completion of this instance of abstraction



g
ś
c
ź
j
g complete 0.515
ArtificialStartTask complete 1.000
c complete 0.507
ArtificialEndTask complete 1.000
j complete 0.510

since the timestamp of ś is set to the timestamp of j

update node/edge metrics

⟨ś, j, ź⟩	ś.NoExecutions++ j.NoExecutions++ ź.NoExecutions++ ś.NoInitializations++ ź.NoTerminations++	(ś, j).NoExecutions++ (j, ź).NoExecutions++	(ś, j).ThroughputTime += 0 (j, ź).ThroughputTime += $\#_{time}(\texttt{c}) - \#_{time}(\texttt{j})$
⟨g⟩	g.NoExecutions++	(ś, g).NoExecutions++ (g, ź).NoExecutions++	(ś, g).ThroughputTime += $\#_{time}(\texttt{g}) - \#_{time}(\texttt{j})$ (g, ź).ThroughputTime += $\#_{time}(\texttt{c}) - \#_{time}(\texttt{g})$
⟨c⟩	c.NoExecutions++	(ś, c).NoExecutions++ (c, ź).NoExecutions++	(ś, c).ThroughputTime += $\#_{time}(\texttt{c}) - \#_{time}(\texttt{j})$ (c, ź).ThroughputTime += 0

since the timestamp of ź is set to the timestamp of c

since the timestamp of ź is set to the timestamp of c

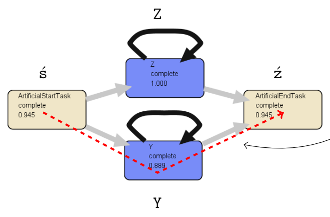
Figure 7.8: Mapping of the pattern jgc at position 1 in the trace $\underline{\mathbf{t_1}}$ = jgcYZfi onto the process model M_{10} and updating the node and edge metrics.

ś, j, g, c, and ź by 1. The number of executions of the edges (ś, j), (ś, g), and (ś, c) is incremented by 1 while their throughput times are increment by $\#_{time}(\mathbf{t}_1(1)) - \#_{time}(\mathbf{t}_1(1)), \#_{time}(\mathbf{t}_1(2)) - \#_{time}(\mathbf{t}_1(1))$, and $\#_{time}(\mathbf{t}_1(3)) - \#_{time}(\mathbf{t}_1(1))$ respectively. Similarly, the number of executions of the edges (j, ź), (g, ź), and (c, ź) is incremented by 1 while their throughput times are incremented by $\#_{time}(\mathbf{t}_1(3)) - \#_{time}(\mathbf{t}_1(1)), \#_{time}(\mathbf{t}_1(3)) - \#_{time}(\mathbf{t}_1(2))$, and $\#_{time}(\mathbf{t}_1(3)) - \#_{time}(\mathbf{t}_1(3))$ respectively. We create a new event e whose activity name is W and its start timestamp $\#_{start}(e) = \#_{time}(\mathbf{t}_1(1))$ and its completion timestamp $\#_{time}(e) = \#_{time}(\mathbf{t}_1(3))$ (Steps 11–12, Algorithm 7.2). We append the event e to the transformed trace $\mathbf{t}'_1$, i.e., $\underline{\mathbf{t}'_1}$ = W (Step 13, Algorithm 7.2). i is updated to 3 (Step 23, Algorithm 7.1).

At the next position 4 (see Figure 7.9), we have a continuous manifestation of the pattern $\mathbf{p}$ = YZ (Step 17, Algorithm 7.1). The process model M_{11} is best associated to $\mathbf{p}$ (Step 19, Algorithm 7.1). We next need to map the pattern $\mathbf{p}$ to the flows in the model M_{11} (Step 21, Algorithm 7.1). The pattern after adding the artificial start/end activities becomes $\mathbf{p}$ = śYZź. Since the first activity in the pattern Y is an abstract activity, the timestamp of the artificial start task is set to the start time of Y. The timestamp of the artificial end task is set to the completion time of the last activity in the pattern, i.e., Z. We now do a guided walk over the graph $G = (V, E)$ (corresponding to M_{11}). For the given pattern śYZź, the mapping terminates with a walk ⟨ś, Y, ź⟩ covering the pattern and an individual activity Z (Step 1,

index	trace	model/ abstraction	pattern with start/end tasks	mapped walks/ activities	transformed trace
4	$\mathtt{j}_1\ \mathtt{g}_2\ \mathtt{c}_3\ \overline{\overset{\downarrow}{\mathtt{Y}}_4\ \mathtt{Z}_5}\ \mathtt{f}_6\ \mathtt{i}_7$	M_{11}/M	śYZź	⟨ś, Y, ź⟩ ⟨Z⟩	WM $\#_{start}(\mathtt{M}) = \#_{start}(\mathtt{Y})$ $\#_{time}(\mathtt{M}) = \#_{time}(\mathtt{Z})$

since a subsequence is being replaced by an abstract activity, we create two timestamps, *start* and *time*, signifying the start/completion of this instance of abstraction. Since the start activity of the pattern is an abstract activity (Y), we set the start time of M to the start time of Y



Z
ś
ź
Y
ArtificialStartTask complete 0.945
Z complete 1.000
ArtificialEndTask complete 0.945
Y complete 0.889

since the timestamp of ś is set to the start timestamp of Y

update node/edge metrics

⟨ś, Y, ź⟩

ś.NoExecutions++
Y.NoExecutions++
ź.NoExecutions++

(ś, Y).NoExecutions++
(Y, ź).NoExecutions++

(ś, Y).ThroughputTime += 0
(Y, ź).ThroughputTime += $\#_{time}(\mathtt{Z}) - \#_{time}(\mathtt{Y})$

since the timestamp of ź is set to the completion timestamp of Z

ś.NoInitializations++
ź.NoTerminations++

Y.ThroughputTime += $\#_{time}(\mathtt{Y}) - \#_{start}(\mathtt{Y})$

since Y is an abstract activity, its throughput time is incremented by the difference between the completion and start time of this instance

⟨Z⟩

Z.NoExecutions++

(ś, Z).NoExecutions++
(Z, ź).NoExecutions++

(ś, Z).ThroughputTime += $\#_{start}(\mathtt{Z}) - \#_{start}(\mathtt{Y})$
(Z, ź).ThroughputTime += 0

since the timestamp of ś is set to the start timestamp of Y

Z.ThroughputTime += $\#_{time}(\mathtt{Z}) - \#_{start}(\mathtt{Z})$

since Z is an abstract activity, its throughput time is incremented by the difference between the completion and start time of this instance

Figure 7.9: Mapping of the pattern YZ at position 4 in the trace $\underline{\mathbf{t}_1}$ = jgcYZfi onto the process model M_{11} and updating the node and edge metrics.

Algorithm 7.2). We update the edge/node metrics corresponding to the walk (flow) and the activities not mapped to any walk (Steps 2–7, Algorithm 7.2). The number of executions of the nodes ś, Y, Z, and ź is incremented by 1. Since the activities Y and Z are abstract activities, we update the node throughput times of these two nodes. Recall that an instance of an abstract activity captures a subsequence of events. The start time of an abstract activity is captured under the attribute $\#_{start}$ while the completion time is captured in $\#_{time}$. The node throughput times of Y and Z are incremented by $\#_{time}(\mathbf{t}_1(4)) - \#_{start}(\mathbf{t}_1(4))$ and $\#_{time}(\mathbf{t}_1(5)) - \#_{start}(\mathbf{t}_1(5))$ respectively.

The number of executions of the edges (ś, Y), (Y, ź), (ś, Z), and (Z, ź) is incremented by 1. Since Y and Z are abstract activities, the edge throughput times of the incoming edges of abstract activities are updated considering their start timestamps while the throughput times of their outgoing edges are updated considering the completion timestamps. In other words, the throughput times of the edges (ś, Y) and (ś, Z) are incremented by 0 and $\#_{start}(\mathbf{t}_1(5)) - \#_{start}(\mathbf{t}_1(4))$ while that of the edges (Y, ź) and (Z, ź) are incremented by $\#_{time}(\mathbf{t}_1(5)) - \#_{time}(\mathbf{t}_1(4))$ and 0 respectively. We create a new event e whose activity name is M and set its start timestamp to that of the start time of $\texttt{Y}$ and the completion timestamp to that of the completion time of $\texttt{Z}$, i.e., $\#_{start}(e) = \#_{start}(\mathbf{t}_1(4))$ and $\#_{time}(e) = \#_{time}(\mathbf{t}_1(5))$ (Steps 11–12, Algorithm 7.2). We append the event e to the transformed trace $\mathbf{t}'_1$, i.e., $\underline{\mathbf{t}'_1} = \texttt{WM}$ (Step 13, Algorithm 7.2). i is updated to 5 (Step 23, Algorithm 7.1).

In the next iterations, the activities $\texttt{f}$ and $\texttt{i}$ at positions 6 and 7 do not belong to this level. We append them to the transformed trace and the final transformed trace is $\underline{\mathbf{t}'_1} = \texttt{WMfi}$, which is at level 0 (Steps 13–15, Algorithm 7.1). Proceeding further, we process all the traces in the event log.

We now process the event log at the top-level (Step 30, Algorithm 7.1). Since the top-level model contains artificial start/end tasks, we add an artificial start/end task event at the beginning and end of each of the traces in the top-level event log. The timestamp of the artificial start event is set to the timestamp of the first event in the trace while that of the artificial end event is set to the last event in the trace. Note that this is unlike the earlier iterations where we added the artificial start/end tasks for each pattern manifestation. Some of the common execution patterns at this level are $P_0 = \{\texttt{\'sW}, \texttt{\'sWM}, \texttt{\'sWMf}, \texttt{\'sWf}, \texttt{i\'z}, \dots\}$. Consider the trace $\underline{\mathbf{t}_1} = \texttt{\'sWMfi\'z}$.

At position 1, there is a continuous manifestation of the pattern $\texttt{\'sWMf}$. Upon mapping the pattern onto the top-level process model, we get a walk $\langle \texttt{\'s}, \texttt{W}, \texttt{M} \rangle$ and an individual activity $\texttt{f}$ covering the pattern (Step 1, Algorithm 7.2). The number of executions of the nodes $\texttt{\'s}, \texttt{W}, \texttt{M}$, and $\texttt{f}$ is incremented by 1 (Steps 2–10, Algorithm 7.2). The throughput times of the abstract nodes $\texttt{W}$ and $\texttt{M}$ are incremented by $\#_{time}(\mathbf{t}_1(2)) - \#_{start}(\mathbf{t}_1(2))$ and $\#_{time}(\mathbf{t}_1(3)) - \#_{start}(\mathbf{t}_1(3))$. The number of executions of the edges $(\texttt{\'s}, \texttt{W})$ and $(\texttt{W}, \texttt{M})$ is incremented by 1. Since the individual activity $\texttt{f}$ is connected to $\texttt{W}$, we increment the number of executions of the edge $(\texttt{W}, \texttt{f})$ by 1. The throughput times of the edges $(\texttt{\'s}, \texttt{W}), (\texttt{W}, \texttt{M})$, and $(\texttt{W}, \texttt{f})$ are incremented by 0, $\#_{start}(\mathbf{t}_1(3)) - \#_{time}(\mathbf{t}_1(2))$ and $\#_{time}(\mathbf{t}_1(4)) - \#_{time}(\mathbf{t}_1(2))$ respectively. In the next iteration, at position 5, there exists a continuous manifestation of the pattern $\texttt{i\'z}$. There exists a walk $\langle \texttt{i}, \texttt{\'z} \rangle$ in the top-level process model (Step 1, Algorithm 7.2). We update the number of executions of the nodes $\texttt{i}$, $\texttt{\'z}$, and the edge $(\texttt{i}, \texttt{\'z})$ by 1. The throughput time of the edge $(\texttt{i}, \texttt{\'z})$ is incremented by 0 (since the timestamp of $\texttt{\'z}$ is the same as that of $\texttt{i}$. Since this pattern is mapped to the same model as the previous pattern, we look back into the trace and identify the events corresponding to the incoming edges to the first activity $\texttt{i}$ of this pattern (Step 15, Algorithm 7.2). The incoming nodes correspond to $\texttt{M}$ and $\texttt{f}$. Since we find both these activities in the look-back, we update the number of executions of the edges $(\texttt{M}, \texttt{i})$ and $(\texttt{f}, \texttt{i})$ by 1 while their throughput times are incremented by $\#_{time}(\mathbf{t}_1(5)) - \#_{time}(\mathbf{t}_1(3))$ and $\#_{time}(\mathbf{t}_1(5)) - \#_{time}(\mathbf{t}_1(4))$ respectively. In this fashion, the performance metrics are computed.

In the next section, we suggest techniques that can assist in discovering interesting diagnostic insights from these KPIs.

7.4 Annotating Fuzzy Maps and Diagnostics

Business analysts can exploit the KPIs estimated during replay and analyze their operational processes. These KPIs provide rich diagnostic information in identifying the bottlenecks in a process, i.e., the activities and/or flows that hinder the performance of a process. An intuitive visualization and highlighting of problematic regions in a process can go a long way in assisting analysts in diagnosing the issues quickly. Though different visualization means such as creating tables and charts can be considered, they alienate the process perspective, which the analysts are most interested in. Therefore, a simple and straightforward means is to annotate the process models with the relevant metrics. This can be augmented with some color coding based on certain criteria (defined later in this section) to highlight the regions of interest such as bottlenecks.

Figure 7.10 depicts the annotation of a node and edges of a Fuzzy process map with performance metrics. The basic visualization and exploration features of Fuzzy models are retained for performance visualization. For example, as discussed in Chapter 6, abstract nodes in a performance annotated process map are colored in blue to differentiate them from primitive nodes, the width of an edge indicates the significance, etc. However, we enrich the process map with additional information. As depicted in Figure 7.10, a process node captures five KPIs:

- the number of times the node is the initial activity of the process for the cases in the event log (labeled `INIT`),
- the number of times the node is the termination activity of the process for the cases in the event log (labeled `TERM`),
- the number of times the node is executed in the event log (labeled `COUNT`),
- the number of times the execution of the node is skipped in the event log (labeled `SKIP`), and
- the average throughput (execution) time of the node.

It is important to note that the nodes/activities in the model are considered to be atomic. Therefore, only abstract nodes have an average throughput time, which signifies the average time spent by the cases in the event log in the subprocesses captured underneath that abstraction. Edges in a process map are annotated with the number of executions. Additional information on the performance metrics can be provided through an interactive visualization, e.g., upon selecting a node, additional information such as the total throughput time, minimum, maximum, and standard deviation of the throughput time can be provided. Similarly, upon selection of an edge, one can provide information such as the minimum, maximum, average, and standard deviation of the throughput time of the edge, the total throughput time, and the number of times the edge is skipped.

We can further enrich the annotation of a process map with color coding to highlight the performance of entities (nodes/edges). The basic idea in performance coloring is to have two thresholds, a *min* and *max* for a metric such as the average throughput time, for each entity in a process map. If the average value of a metric of an entity is

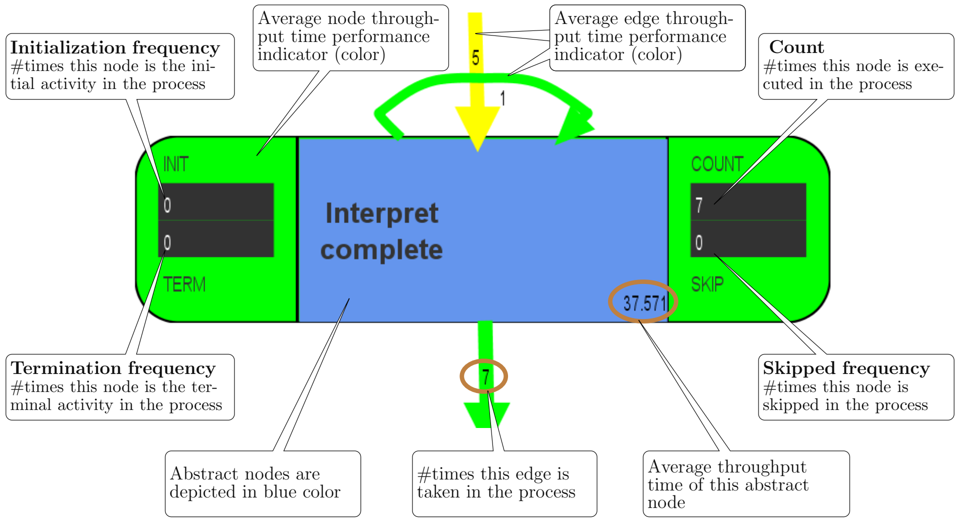

Initialization frequency
#times this node is the initial activity in the process
Average node throughput time performance indicator (color)
5
Average edge throughput time performance indicator (color)
Count
#times this node is executed in the process
1
INIT
0
0
TERM
Interpret complete
COUNT
7
0
SKIP
37.571
7
Termination frequency
#times this node is the terminal activity in the process
Skipped frequency
#times this node is skipped in the process
Abstract nodes are depicted in blue color
#times this edge is taken in the process
Average throughput time of this abstract node

Figure 7.10: Annotating a node and edge in a process model with KPIs for visual inspection.

below the *min* threshold, we color an entity with green, if the average is between the *min* and *max* thresholds, we color it with yellow, and if the average is above the *max* threshold, we color it with red. Entities colored in red are potential bottlenecks in the process and require critical inspection. We suggest three criteria for consideration as the *min* and *max* thresholds of entities in a process map.

- *Model Specific Averages:* In this strategy, the threshold for an entity in a process map is defined based on the global averages of the process model in which it manifests. More specifically, for a chosen metric such as the throughput time, we compute the average metric value for each of the models in the process map. All nodes in the model carry the same min and max thresholds, which are defined as percentages of the average metric value of that model. In other words, for any model M, let $NodeAvg_M$ denote the average of the metric value of all the nodes in the model. For some chosen percentages $\delta_{\text{min}} \in (0,1)$ and $\delta_{\text{max}} \in (0,1)$, the *min* and *max* thresholds for all nodes v in the model M are defined as

$$min = (1 - \delta_{\text{min}}) \times NodeAvg_M$$
$$max = (1 + \delta_{\text{max}}) \times NodeAvg_M$$

  The color coding of a node is then defined based on these *min* and *max* thresholds. The basic intuition behind this coloring scheme is that we consider nodes whose average metric value is significantly higher than that of the models' average as potential bottlenecks (where the significance bounds are provided by the user in terms of acceptable percentages with respect to the model average value). The edge thresholds can be defined in a similar manner.

- *Large Deviations:* Considering fixed thresholds for all nodes/edges in a model might not always be appropriate, especially in scenarios where the process execution time is dominated by a few activities. For example, in the insurance claim process described in Section 5.1, the predominant time consuming activities are the ones pertaining to the checks to be done in order to determine the validity of a claim. In such scenarios, the performance insights obtained using global model averages are too obvious for analysts. Therefore, we propose an alternative strategy where the thresholds are defined more locally (specific to each entity). In this strategy, we consider entities in a model with large deviations as potential bottlenecks. For example, if a task has a large deviation with respect to its execution time, it indicates that for some cases, the task can be completed quickly but for others it takes longer. It provides an opportunity for analysts to investigate the reasons for such a phenomenon and thereby improve their processes. In this strategy, unlike the model specific averages, each node/edge has a different threshold based on its average metric value. For some chosen percentages $\delta_{\min} \in (0,1)$ and $\delta_{\max} \in (0,1)$ and $\delta_{\max} > \delta_{\min}$, The *min* and *max* thresholds for the standard deviation metric value of an entity are defined as

$$\begin{aligned} min &= \delta_{\min} \times avg \\ max &= \delta_{\max} \times avg \end{aligned}$$

where *avg* is the average metric value of the entity. In other words, nodes/edges whose execution time standard deviation is less than their corresponding min threshold are colored green, while nodes/edges whose execution time standard deviation is between their min and max thresholds are colored yellow, and nodes/edges whose execution time standard deviation is above their max threshold are colored red.

- *Benchmark Thresholds:* The third strategy for performance coloring is based on user defined benchmark thresholds for each activity/edge. Such thresholds can emanate from standard/best practices or from simulation models. For example, in the insurance claim process, the organization might specify the expected execution time of tasks such as the high insurance check takes between 15 and 20 minutes. If upon replaying an event log, the average metric values exceed this range, they can be highlighted as potential bottlenecks.

7.5 Experiments and Discussion

In this section, we apply the proposed approach for performance analysis on the digital copier example. We have considered a log $\mathcal{L}$ containing 100 traces, 76 event classes, and 40,995 events. 51 traces pertain to print job requests while the remaining 49 traces pertain to copy/scan jobs. We first discover a process map using this event log. The process map corresponds to the one depicted in Figures 6.25, 6.26, and 6.27. Figure 7.11 depicts the tree representation of the process map. This process map contains four hierarchical levels. The event log $\mathcal{L}$ is at the finest granularity, i.e., level 3. We replay the log onto this process map and compute the performance metrics.

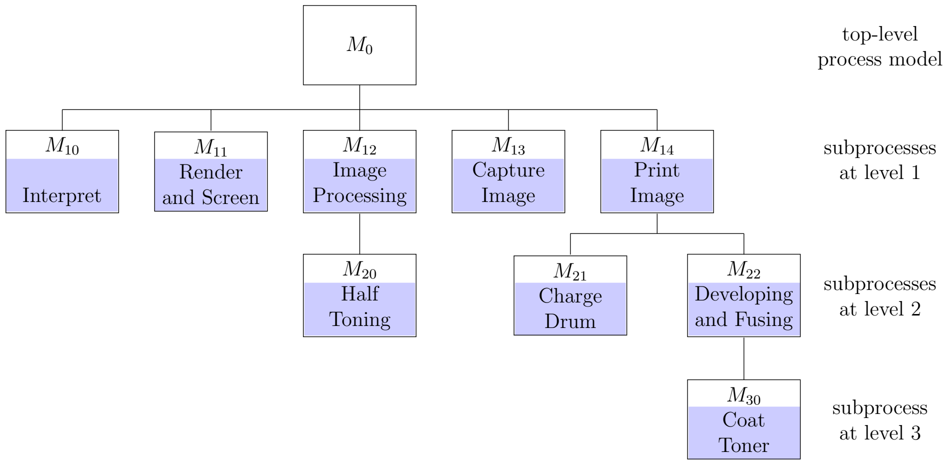


Figure 7.11: Process map of the digital copier example (discussed in Section 6.5) visualized as a tree of process models. Subprocesses are annotated with their corresponding abstract activity. The process map of the digital copier can be inspected in Figures 6.25, 6.26, and 6.27.

Figure 7.12 depicts the top-level process model annotated with performance measures. As indicated in the figure, all cases start at the activity Job–start (this is reflected in the KPI No.Initializations=100 for this node). Similarly, all cases terminate at Job–complete. The coloring of nodes and edges in Figure 7.12 is based on model specific averages with tolerance thresholds of ±10%. Nodes/edges whose average throughput time is lower than 90% of the model's average node/edge throughput time are colored green while those with average throughput time greater than 110% are colored red. Nodes/edges with average throughput time between 90% and 110% of the model's average node/edge throughput time are colored yellow. The average node throughput time for this model is 505.925 time units while the average edge throughput time is 21877.607 time units. The relatively high average edge throughput time for this model is solely due to the queuing of jobs before they are processed[2], i.e., between Job–start and Copy/Scan–complete and between Job–start and Remote Print–complete. This is reflected in these edges acquiring a red color indicating that their average throughput times are way above the model's average.

Nodes that are colored blue at the center are abstract nodes. As mentioned in Section 7.2, node throughput times are only defined for abstract nodes (as they capture a subprocess underneath). There are five abstract nodes in this model, viz., Interpret–complete, Render and Screen–complete, Capture Image–complete, Image Processing–complete, and Print Image–complete. Among these, the nodes Capture Image–complete and Print Image–complete are colored red because their average throughput times, 888.837 and 7617.686 time units respectively, are at least 10% over and above the model's average node throughput time (i.e., 505.925). Furthermore, as mentioned in Chapter 3, the rendering and screening of pages is relatively

[2]In the copier, a new job request cannot be processed until the current job is finished.

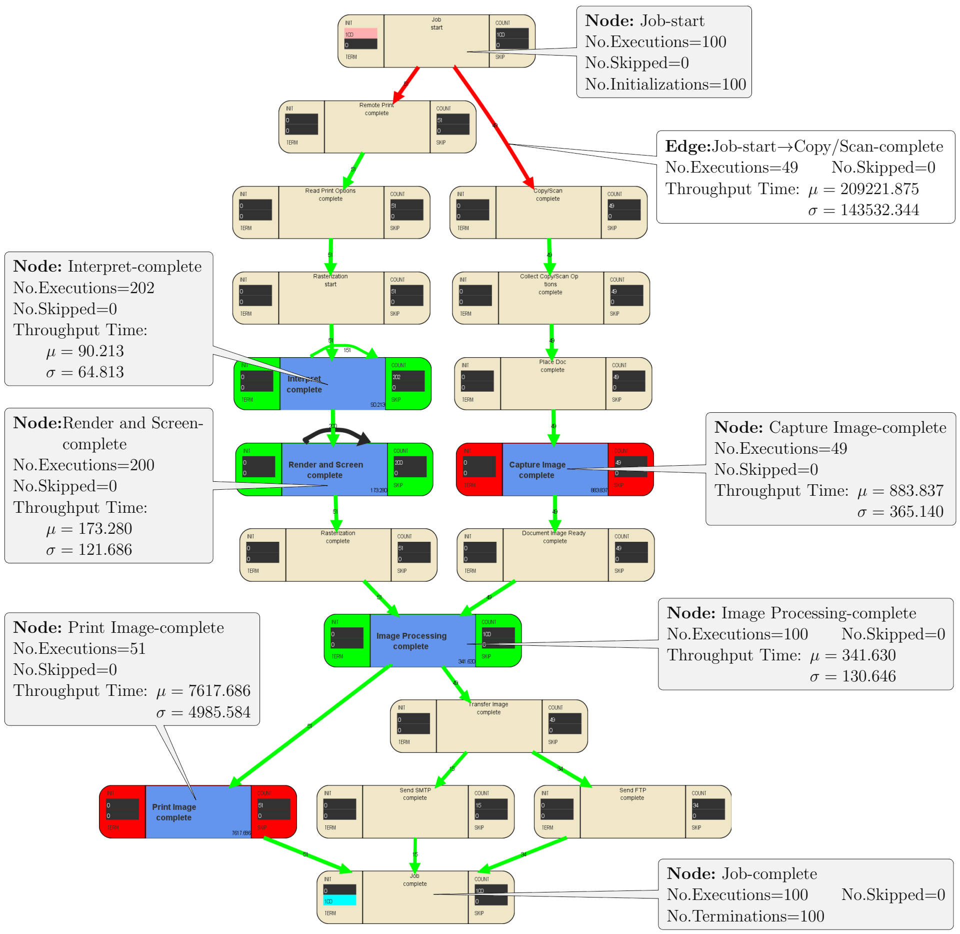


Figure 7.12: The top-level process model of the digital copier example showing performance related information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±10%. The model's average node and edge throughput times are 505.925 and 21877.607 time units respectively.

more time consuming than the interpretation of pages. This is reflected in the average node throughput times of Render and Screen–complete and Interpret–complete. From the figure, we can conclude that Capture Image–complete and Print Image–complete are the major bottleneck activities. Likewise, the colored edges indicate that the queuing of jobs between the submission and the start of processing is causing delays. These insights can be used to improve the process.

Figure 7.13 depicts the annotated subprocess corresponding to the abstract node Interpret–complete. This subprocess is invoked for print job requests and pertains to the interpretation of document pages submitted for print. Three types of interpretation are supported, viz., post script, page control language, and unformatted text. All instances of this subprocess start at the activity Interpretation–start and

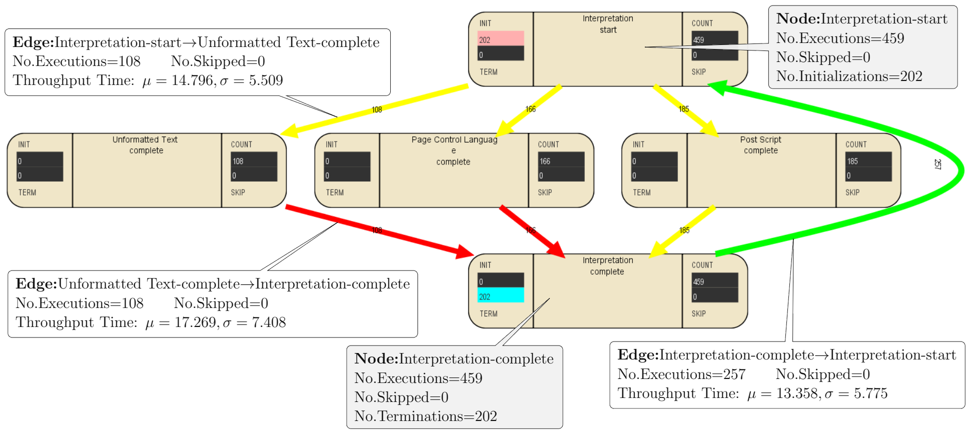

Edge:Interpretation-start→Unformatted Text-complete
No.Executions=108 No.Skipped=0
Throughput Time: $\mu = 14.796, \sigma = 5.509$
Interpretation start
Node:Interpretation-start
No.Executions=459
No.Skipped=0
No.Initializations=202
Unformatted Text complete
Page Control Language complete
Post Script complete
108
166
185
257
Interpretation complete
Edge:Unformatted Text-complete→Interpretation-complete
No.Executions=108 No.Skipped=0
Throughput Time: $\mu = 17.269, \sigma = 7.408$
Node:Interpretation-complete
No.Executions=459
No.Skipped=0
No.Terminations=202
Edge:Interpretation-complete→Interpretation-start
No.Executions=257 No.Skipped=0
Throughput Time: $\mu = 13.358, \sigma = 5.775$
INIT
TERM
COUNT
SKIP

Figure 7.13: The subprocess corresponding to the abstract activity Interpret–complete showing performance related information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±10%. The model's average edge throughput time is 15.625 time units.

terminate at Interpretation–complete. Note that although the number of print job requests is 51, the number of executions of Interpretation–start and Interpretation–complete is 459. This is due to the fact that a document can have multiple pages and each page in the document needs to be interpreted separately. Furthermore, note that the number of initializations/terminations in this subprocess is 202. Recall from Figure 3.3 that this subprocess can be executed in parallel with the rendering and screening subprocess after at least one page has been interpreted. A continuous interpretation of pages without an interleaved execution of rendering and screening is captured within the same instance of invocation of this subprocess (reflected in the loop construct; there are 257 executions of the edge between Interpretation–complete and Interpretation–start).

This subprocess does not contain any abstract nodes. So this process does not have an average node throughput time. The average edge throughput time of this subprocess is 15.625 time units. Figure 7.13 depicts the coloring of edges using the model's average edge throughput time with tolerance limits of ±10%. Since the average throughput times of the edges (Unformatted Text–complete, Interpretation–complete) and (Page Control Language–complete, Interpretation–complete), 17.269 and 17.675 time units respectively, are at least 10% over and above the model's average edge throughput time (i.e., 15.625), they are colored red. In contrast, as the average throughput time of the edge (Interpretation–complete, Interpretation–start) (i.e., 13.358) is at least 10% less than the model's average edge throughput time, it is colored green.

Figure 7.14 depicts the annotated subprocess corresponding to the abstract activity Coat Toner–complete at the third level of hierarchy. This is a subprocess of the Print Image–complete subprocess. Again the colors of nodes and edges show where most time is lost.

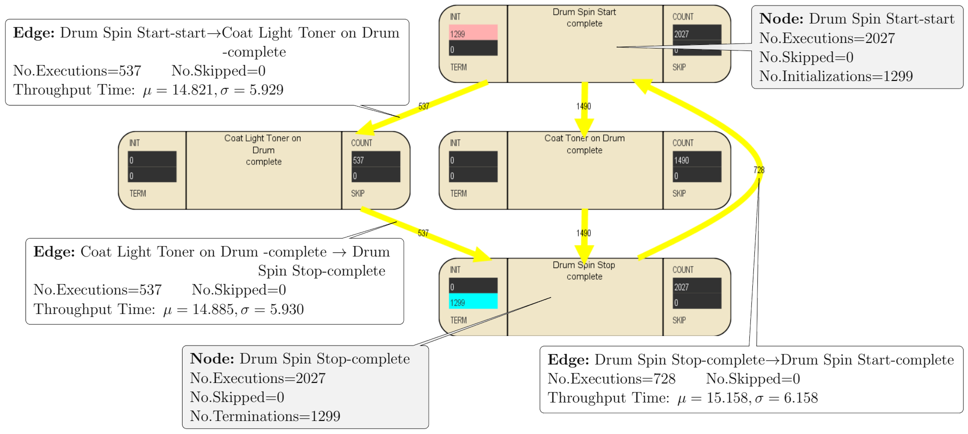

Edge: Drum Spin Start-start→Coat Light Toner on Drum -complete
No.Executions=537 No.Skipped=0
Throughput Time: $\mu = 14.821, \sigma = 5.929$
Drum Spin Start complete
INIT 1299 0 TERM
COUNT 2027 0 SKIP
Node: Drum Spin Start-start
No.Executions=2027
No.Skipped=0
No.Initializations=1299
537
1490
728
Coat Light Toner on Drum complete
INIT 0 0 TERM
COUNT 537 0 SKIP
Coat Toner on Drum complete
INIT 0 0 TERM
COUNT 1490 0 SKIP
Edge: Coat Light Toner on Drum -complete → Drum Spin Stop-complete
No.Executions=537 No.Skipped=0
Throughput Time: $\mu = 14.885, \sigma = 5.930$
537
1490
Drum Spin Stop complete
INIT 0 1299 TERM
COUNT 2027 0 SKIP
Node: Drum Spin Stop-complete
No.Executions=2027
No.Skipped=0
No.Terminations=1299
Edge: Drum Spin Stop-complete→Drum Spin Start-complete
No.Executions=728 No.Skipped=0
Throughput Time: $\mu = 15.158, \sigma = 6.158$

Figure 7.14: The subprocess corresponding to the abstract activity Coat Toner–complete annotated with performance information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±10%. The model's average edge throughput time is 14.959 time units.

As discussed earlier, model specific averages might not always lead to useful insights. This is reflected in Figure 7.12 where the model's average edge throughput time is dominated by just two edges (cf. red output arcs of Job–Start node). This resulted in all other edges acquiring a green color. To alleviate this, we use the large deviations strategy for performance coloring of the digital copier process map. Figure 7.15 depicts the top-level process model annotated with performance measures with the coloring of nodes/edges based on the criteria of large deviations. We used the thresholds of $\delta_{\text{min}} = 0.3$ and $\delta_{\text{max}} = 0.6$. In other words, if the ratio between the standard deviation and the average throughput time of a node/edge is less than 0.3, it is colored green; if the ratio is between 0.3 and 0.6, it is colored yellow; otherwise, it is colored red. This results in additional insights on the process execution. For example, the nodes Interpret–complete and Render and Screen–complete now acquire a red color. This is due to the fact that these nodes exhibit a large variance among the instances of execution, e.g., the standard deviation of Interpret–complete is 64.813, which is approximately 72% of the node's average throughput time; similarly, the standard deviation of Render and Screen–complete is 121.686, approximately 70% of the node's average throughput time. Likewise, the node Capture Image–complete, which has a relatively higher average throughput time when compared to the model's average node throughput time acquires a yellow color based on variance (its standard deviation, which is 365.140, is 41% of its average throughput time).

Figure 7.16 depicts the annotated process model of the subprocess corresponding to the abstract activity Interpret–complete. The ratio between the standard deviation and the average throughput time of all edges in this model lies between 0.3 and 0.6. Hence, all edges are colored yellow in this figure.

Figures 7.12–7.16 demonstrate that performance measures estimated by replay-

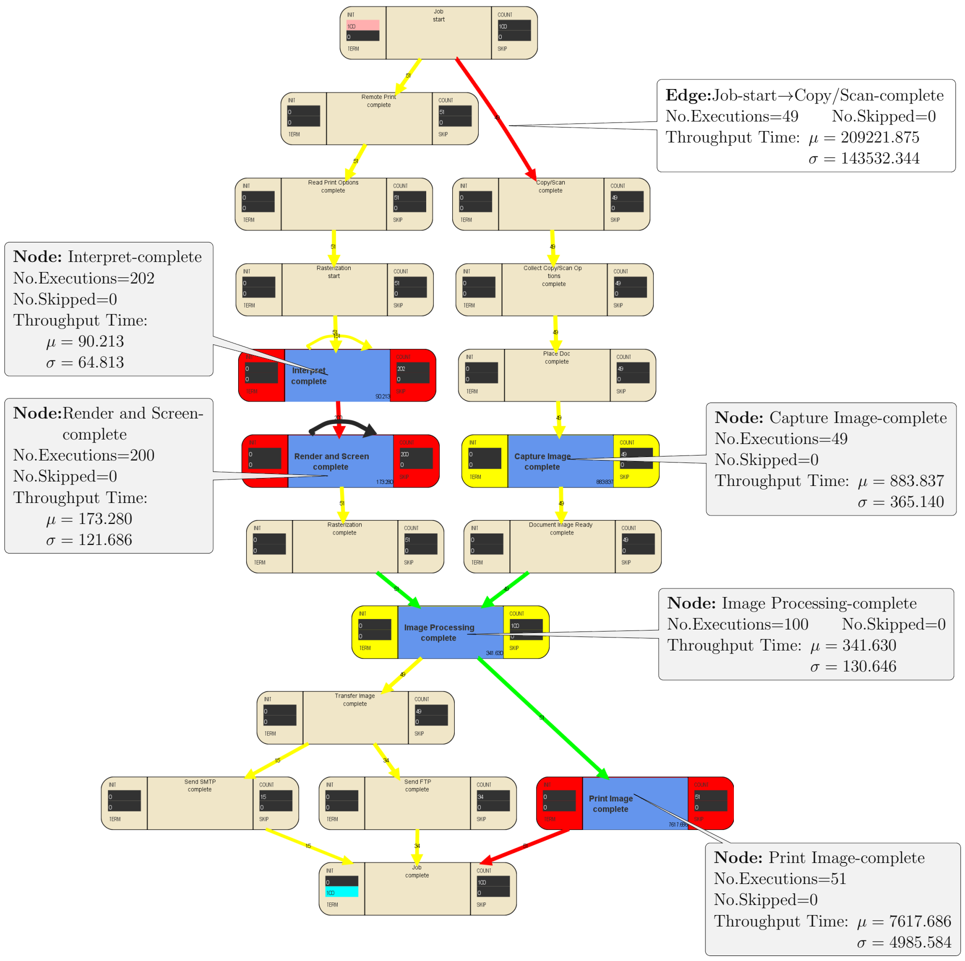
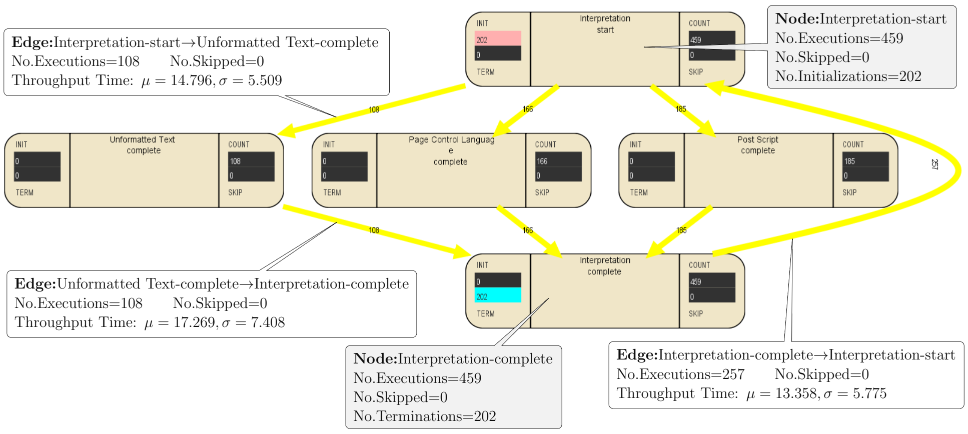


Figure 7.15: The top-level process model of the digital copier example annotated with performance information. The coloring of nodes/edges is based on variance (large deviations) with tolerance limits of $\delta_{\min} = 0.3$ and $\delta_{\max} = 0.6$.

ing an event log onto a process map can be used to identify potential bottlenecks. Such insights can help us in process redesign and optimization efforts.



Interpretation start
INIT
202
0
TERM
COUNT
459
0
SKIP
Node:Interpretation-start
No.Executions=459
No.Skipped=0
No.Initializations=202
Edge:Interpretation-start→Unformatted Text-complete
No.Executions=108 No.Skipped=0
Throughput Time: μ = 14.796, σ = 5.509
108
166
185
257
Unformatted Text complete
108
Page Control Language complete
166
Post Script complete
185
Interpretation complete
459
202
Edge:Unformatted Text-complete→Interpretation-complete
No.Executions=108 No.Skipped=0
Throughput Time: μ = 17.269, σ = 7.408
Node:Interpretation-complete
No.Executions=459
No.Skipped=0
No.Terminations=202
Edge:Interpretation-complete→Interpretation-start
No.Executions=257 No.Skipped=0
Throughput Time: μ = 13.358, σ = 5.775

Figure 7.16: Annotation of the subprocess corresponding to the abstract activity Interpret–complete with performance measures. The coloring of nodes/edges is based on variance (large deviations) with tolerance limits of $\delta_{\min} = 0.3$ and $\delta_{\max} = 0.6$.

7.6 Conclusions

In this chapter, we showed that it is possible to enrich process maps with performance information. Annotating process maps with KPI-based information provides useful insights and helps the analyst to identify bottlenecks in the process under consideration. We proposed an approach of replaying an event log onto a process map and estimating key performance measures. The proposed approach is amenable to process maps with multiple levels of hierarchy and can accustom an event log at any level of hierarchy with respect to the process map. Furthermore, we suggested three criteria for the automatic identification of potential bottlenecks based on the computed performance measures.

Chapter 8
Trace Alignment

In the previous chapters, we looked at techniques that deal with less-structured processes. Although these techniques result in improved comprehensibility of the discovered processes, analyzing the models for diagnostic purposes, especially to answer the questions raised in Chapter 1, still requires considerable effort. In this chapter, we advocate that process diagnostics can be assisted through alternative means of *careful inspection of the event log by grouping and aligning the traces found in the event log*. We propose an approach, called *trace alignment*, which is inspired from biological sequence alignment [260]. *Trace alignment consists of two steps: first, we group similar traces in clusters using the agglomerative hierarchical clustering technique as discussed in Chapter 4; second, we visualize these clusters by aligning the traces.* By aligning traces we can see the common and frequent behavior, and distinguish this from the exceptional behavior. We further show that trace alignment can be used to answer a variety of diagnostic questions and that it is a welcome addition to the repertoire of process mining techniques. More specifically, trace alignment can assist in answering the following diagnostic questions raised in Chapter 1:

- *What is the most common (likely) process behavior that is executed?*
- *Where do process instances deviate and what do they have in common?*
- *Are there any common patterns of execution in the traces?*
- *What are the contexts in which an activity or a set of activities is executed in a event log?*
- *What are the process instances that share/capture a desired behavior either exactly or approximately?*
- *Are there particular patterns (e.g., milestones, concurrent activities, etc.) in the process?*

The remainder of this chapter is organized as follows. Related work is presented in Section 8.1. Section 8.2 introduces the concept of trace alignment and discusses the techniques for finding alignments. We propose a framework for finding alignments over a bag of traces in Section 8.3 and present techniques for refining alignments to improve alignment quality in Section 8.4. Section 8.5 discusses the computational complexity of performing alignments. The experimental results and scalability issues are discussed in Section 8.6. We provide an outlook on some of the opportunities and challenges in trace alignment for future research in Section 8.7. Finally, Section 8.8 concludes the chapter.

8.1 Related Work

Trace alignment is a log visualization technique that assists in uncovering interesting insights in process executions. Dotted chart analysis [207] is one of the most commonly used log visualization technique. Dotted chart analysis, analogous to Gantt charts, present a “helicopter view” of the event log and assist in analyzing process performance by depicting process events in a graphical way. Dotted charts primarily focus on the time dimension of events. The dotted chart analysis also computes some metrics for performance such as the minimum, maximum, and average interval between events. A business analyst needs to *manually* investigate the dotted chart to identify any potential performance issues. For logs with many activities, the manual inspection and comprehension of the dotted chart becomes cumbersome and often infeasible to identify interesting patterns. Trace alignment alleviates this problem, by finding those patterns automatically and presenting them to the user. In the parlance of dotted chart analysis, trace alignment considers the *logical relative* time perspective of the event log. Furthermore, it would be simple and a natural extension to project the performance metrics proposed in [207] onto the aligned traces.

Stream scope visualization [91] is a trace visualization technique that is based on the event class correlations. Using stream scope visualization, patterns of co-occurring events can easily be recognized by their vicinity. However, stream scope visualization is restricted in that it visualizes each trace separately and does not provide a holistic view of the event log. In contrast, trace alignment enables the visualization of multiple traces at a time and is able to uncover common execution patterns within and across traces.

One of the applications of trace alignment is in uncovering deviations between anomalous and normative traces. Conformance checking aims at detecting inconsistencies/deviations between a process model (that captures the expected behavior) and its corresponding execution log [191]. Several techniques for conformance checking exist [3, 4, 191, 237]. However, conformance checking has inherent limitations in its applicability, especially for diagnostic purposes. Firstly, it assumes the existence of a process model or a set of rules. However, in reality, process models are either not present or if present are incorrect or outdated (their quality typically leaves much to be desired). One can argue that process models can be discovered from event logs and conformance checking be applied on the discovered models. However, this approach is not suitable for the analysis of highly complex and/or flexible processes, the class of models which most of the real-life logs fall into and where the discovered models are “spaghetti-like”. Even in cases where the process models are available, it is difficult to look inside of the processes to identify and locate problems, especially with models that are large. Trace alignment is complementary to this approach in that it highlights the deviations by analyzing the raw event traces (avoiding the need for process models).

Declarative modeling is a paradigm that lets users model a business process through a set of rules. In this paradigm, users typically specify only the mandatory and undesired behavior. Many formalisms exist for specifying the rules/constraints, e.g., in Declare [169–171], business processes are described using Linear Temporal Logic (LTL) constraints. Analysts can verify the satisfaction of these constraints in event logs [142, 230]. However, this approach too suffers from the limitations mentioned above: (a) it is not a trivial task to elicit all the constraints and (b) techniques to mine Declare models from event logs generate too many constraints [143].

Another application of trace alignment is in uncovering common patterns of execution. As discussed in Chapter 3, common execution patterns such as tandem arrays and maximal repeats can be discovered in linear time. However, the patterns uncovered are atomic and the dependencies/correlations between patterns need to be discovered separately; in other words, the contexts of their manifestation is lost and needs to be established separately. Common patterns of interaction between activities can also be captured using sequence diagrams [110]. However, this approach too suffers from the limitation that the patterns discovered are atomic, thus mandating the need for establishing correlations/dependencies between patterns separately. In addition, the number of sequence patterns uncovered can be enormous. In contrast, trace alignment provides a holistic view of the traces thereby enabling the discovery of both the common execution patterns and their contexts and (long-range) dependencies.

Trace alignment is largely inspired from Multiple Sequence Alignment (MSA) [84, 260], often used in bioinformatics. However, there are challenges in adapting these techniques for trace alignment. Alignment of biological sequences typically happen over homologous sequences and with little variation in length [163]. However, traces in an event log in process mining need not stem from a coherent set of cases and can be of different lengths. Variation in lengths can occur due to variation in execution paths of the instances and due to manifestation of process model constructs such as the choice and/or loop constructs. In biological sequence alignment, there are standard scoring matrices for substitution that are derived based on physio-chemical properties of the amino acids. Insertion/deletion operations are primarily considered either with a constant gap-score (or penalty) or as an affine function. In contrast, indel and substitution scores for trace alignment need to be context-sensitive and either have to be derived automatically from the event log or provided by the domain experts (see Sections 4.3 and 4.4). Furthermore, biological sequences deal with an alphabet size of either 4 (for four nucleic acids) or 20 (for amino acids). However, the number of distinct activities (event classes) in a typical process mining log can be of the order of a few tens and even hundreds. This adds to the complexity of deriving good scoring matrices and aligning traces. Inspired by MSA techniques [47, 67, 68], this chapter provides techniques for trace alignment.

8.2 Aligning Traces

In this section, we first formally define what trace alignment is and later discuss techniques for finding optimal alignments.

Definition 8.1 (Trace Alignment). Trace alignment of an event log $\mathcal{L} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n]$ over $\mathcal{A}^+$ is defined as a mapping of the bag of traces in $\mathcal{L}$ to another bag of traces $\overline{\mathcal{L}} = [\overline{\mathbf{t}_1}, \overline{\mathbf{t}_2}, \ldots, \overline{\mathbf{t}_n}]$ where each $\overline{\mathbf{t}_i} \in (\mathcal{A} \cup \{-\})^+$ for $1 \leq i \leq n$ and

- there is an $m \in \mathbb{N}$, called the *length* of the alignment, such that $|\overline{\mathbf{t}_1}| = |\overline{\mathbf{t}_2}| = \ldots = |\overline{\mathbf{t}_n}| = m$,
- $\overline{\mathbf{t}_i}$ is equal to $\mathbf{t}_i$ after removing all gap symbols "–", and
- there is no $k \in \{1, \ldots, m\}$ such that $\forall_{1 \leq i \leq n}$, $\overline{\mathbf{t}_i}(k) = -$

⌟

An alignment over a bag of traces can be represented by an $n \times m$ rectangular matrix, $A = [a_{ij}]$ $(1 \leq i \leq n, 1 \leq j \leq m)$, over $\mathcal{A}' = \mathcal{A} \cup \{-\}$ where "–" denotes a gap. Figure 8.1 depicts the matrix representation of an alignment of five traces $\mathcal{L} = [$`jgcflebd`, `jgclebdfi`, `jgclebdf`, `jgclfebd`, `jgclefbdi`$]$. The third condition in the definition above implies that no column in A contains only gaps (–). It is imperative to note that there can be many possible alignments for a given bag of traces and that the length of the alignment, m, satisfies the relation $l_{max} \leq m \leq l_{sum}$ where l_{max} is the maximum length of the traces in $\mathcal{L}$ and l_{sum} is the sum of lengths of all the traces in $\mathcal{L}$.

$$\begin{bmatrix}
\mathtt{j} & \mathtt{g} & \mathtt{c} & \mathtt{f} & \mathtt{l} & - & - & \mathtt{e} & \mathtt{b} & \mathtt{d} & - & - \\
\mathtt{j} & \mathtt{g} & \mathtt{c} & - & \mathtt{l} & - & - & \mathtt{e} & \mathtt{b} & \mathtt{d} & \mathtt{f} & \mathtt{i} \\
\mathtt{j} & \mathtt{g} & \mathtt{c} & - & \mathtt{l} & - & - & \mathtt{e} & \mathtt{b} & \mathtt{d} & \mathtt{f} & - \\
\mathtt{j} & \mathtt{g} & \mathtt{c} & - & \mathtt{l} & - & \mathtt{f} & \mathtt{e} & \mathtt{b} & \mathtt{d} & - & - \\
\mathtt{j} & \mathtt{g} & \mathtt{c} & - & \mathtt{l} & \mathtt{e} & \mathtt{f} & - & \mathtt{b} & \mathtt{d} & - & \mathtt{i}
\end{bmatrix}$$

Figure 8.1: A matrix representation of aligned traces.

8.2.1 Pair-wise Trace Alignment

Before we get into the details of aligning a bag of traces, we first consider a special case of trace alignment, where the number of traces to align is 2. Aligning a pair of traces is referred to as *pair-wise* trace alignment. Consider the example of aligning the two traces $\mathbf{t}_1 =$ `abcac` and $\mathbf{t}_2 =$ `acacad`. Figure 8.2 depicts three of the many variants of aligning the two traces.

$$\begin{array}{lccccccc}
\overline{\mathbf{t}_1} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - & - \\
\overline{\mathbf{t}_2} & \mathtt{a} & - & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d}
\end{array}
\qquad
\begin{array}{lcccccc}
\overline{\mathbf{t}_1} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - \\
\overline{\mathbf{t}_2} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d}
\end{array}
\qquad
\begin{array}{lccccccccccc}
\overline{\mathbf{t}_1} & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{a} & \mathtt{c} & - & - & - & - & - & - \\
\overline{\mathbf{t}_2} & - & - & - & - & - & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{c} & \mathtt{a} & \mathtt{d}
\end{array}$$

(a) (b) (c)

Figure 8.2: An example of pairwise trace alignments.

An alignment between a pair of traces, $\mathbf{t}_1$ and $\mathbf{t}_2$, is based on the concept of *edit distance* defined in Chapter 4 and can be considered as a transformation of the trace $\mathbf{t}_1$ to $\mathbf{t}_2$ or vice versa through a set of editing operations applied to one of the traces iteratively. The traces are said to be aligned after the transformation, and can be represented by a rectangular matrix as mentioned earlier. Assuming that $\mathbf{t}_1$ is transformed into $\mathbf{t}_2$, the following edit operations are defined for any column j in the alignment:

- the activity pair $(\mathtt{a}, \mathtt{b})$, $\mathtt{a}, \mathtt{b} \in \mathcal{A}$, denotes the substitution of activity $\mathtt{a}$ in $\mathbf{t}_1$ with activity $\mathtt{b}$ of $\mathbf{t}_2$,
- the activity pair $(\mathtt{a}, -)$ denotes the deletion of activity $\mathtt{a}$ in $\mathbf{t}_1$, and
- the activity pair $(-, \mathtt{b})$ denotes the insertion of activity $\mathtt{b}$ in $\mathbf{t}_1$.

It is important to note that insertion and deletion operations are complementary in that an insertion in one trace can be considered as a deletion in another trace. Henceforth, we refer to insertion and deletion operations as *indel* operations. As discussed in Sections 4.3 and 4.4, *indels* should be sensitive to the context in which the operations are performed. We can consider two notions of context for indels, viz., Indel Right Given Left and Indel Left Given Right (cf. Section 4.4.2), which indicates the insertion of an activity to the immediate right of another activity or to the immediate left of another activity respectively. We consider Indel Right Given Left as the notion of indels for trace alignment.

Furthermore, as discussed in Section 4.4, a score or cost function needs to be defined for the substitution and indel operations to avoid edit operations that do not make sense in a certain context. The substitution score is a function $\mathcal{S} : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$ where $\mathcal{S}(\mathtt{a}, \mathtt{b})$ denotes the score for substitution of activity $\mathtt{a}$ with activity $\mathtt{b}$ for all $\mathtt{a}, \mathtt{b} \in \mathcal{A}$. The Indel Right Given Left score is a function $\mathcal{I}_R : (\mathcal{A} \cup \{-\}) \times (\mathcal{A} \cup \{-\}) \to \mathbb{R}$ where $\mathcal{I}_R(\mathtt{a}, \mathtt{b})$ denotes the score for inserting or deleting activity $\mathtt{b}$ given that the left activity is $\mathtt{a}$ for all $\mathtt{a}, \mathtt{b} \in \mathcal{A}$. $\mathcal{I}_R(-, \mathtt{a}) = \mathcal{I}_R(\mathtt{a}, -) = \mathcal{I}(-, -) = 0$ for all $\mathtt{a} \in \mathcal{A}$.

Let $\overline{\mathbf{t}_1}$ and $\overline{\mathbf{t}_2}$ be the aligned traces of $\mathbf{t}_1$ and $\mathbf{t}_2$ and let m be the length of the alignment. It could be the case that the first activity in $\overline{\mathbf{t}_1}$ or $\overline{\mathbf{t}_2}$ had to be inserted/deleted. The left activity for this case does not exist; so, we define $\overline{\mathbf{t}_1}(0) = \overline{\mathbf{t}_2}(0) = -$. Given $\mathcal{S}$ and $\mathcal{I}_R$, *the score of a pair-wise alignment can be defined as the sum of the scores of the edit operations across all columns in the alignment.* In other words:

$$\text{Score}(\overline{\mathbf{t}_1}, \overline{\mathbf{t}_2}) = \sum_{j=1}^{m} e_j$$

where

$$e_j = \begin{cases} \mathcal{S}(\mathtt{a}, \mathtt{b}) & \text{if } \overline{\mathbf{t}_1}(j) = \mathtt{a} \text{ and } \overline{\mathbf{t}_2}(j) = \mathtt{b} \\ \mathcal{I}_R(\mathtt{a}, \mathtt{b}) & \begin{cases} \text{if } \overline{\mathbf{t}_1}(j) = \mathtt{b}, \overline{\mathbf{t}_1}(j-1) = \mathtt{a} \text{ and } \overline{\mathbf{t}_2}(j) = - \text{ or} \\ \text{if } \overline{\mathbf{t}_1}(j) = -, \overline{\mathbf{t}_2}(j) = \mathtt{b} \text{ and } \overline{\mathbf{t}_2}(j-1) = \mathtt{a} \end{cases} \end{cases}$$

Assuming a *unit score* function where the substitution score function, $\mathcal{S}(\mathtt{a}, \mathtt{b}) = 1$ if $\mathtt{a} = \mathtt{b}$ and $\mathcal{S}(\mathtt{a}, \mathtt{b}) = -1$, otherwise, and an indel score function, $\mathcal{I}_R(\mathtt{a}, \mathtt{b}) = -1$, for all $\mathtt{a}, \mathtt{b} \in \mathcal{A}$, the alignments enumerated in Figure 8.2 have the scores 1, −4, and −9 respectively. A "best" alignment can be considered to be the one with the maximum score. For the above example, this corresponds to Figure 8.2(a). Instead, if we define the indel score function as above but change the substitution score function to $\mathcal{S}(\mathtt{a},\mathtt{b}) = 2$ if $\mathtt{a} = \mathtt{b}$ and $\mathcal{S}(\mathtt{a},\mathtt{b}) = 1$, otherwise, then the alignments enumerated in Figure 8.2 have the scores 5, 5, and −9 respectively. The maximum alignment score is 5 and there are two best alignments corresponding to Figures 8.2(a) and (b). As another example, if we define the substitution score as $\mathcal{S}(\mathtt{a},\mathtt{b}) = 1$ if $\mathtt{a} = \mathtt{b}$ and $\mathcal{S}(\mathtt{a},\mathtt{b}) = -1$, otherwise, and the indel score function as $\mathcal{I}_R(\mathtt{a},\mathtt{b}) = 1$, for all $\mathtt{a},\mathtt{b} \in \mathcal{A}$, the alignments enumerated in Figure 8.2 have the scores 7, −2, and 9 respectively. The maximum score is 9 and the best alignment corresponds to Figure 8.2(c). It is imperative to note that the best scoring alignment is sensitive to the substitution and indel score functions. We adopt the substitution and indel scores derived from an event log as discussed in Section 4.4.

Figure 8.2 depicts just three of the many variants of aligning the two traces $\mathbf{t}_1$ and $\mathbf{t}_2$. In fact, the number of possible alignments for two traces of length l is $\approx (1+\sqrt{2})^{2l+1} l^{-1/2}$ [260], e.g., for two traces of length 100, the number of possible alignments is approximately 10^{77}. Therefore, it is infeasible to enumerate all possible alignments (even for moderate values of l), find their scores, and identify the best alignment. In the next section, we discuss a method of finding a best alignment.

8.2.2 How to Compute Alignments?

Needleman and Wunsch [159] have proposed a dynamic programming algorithm for finding the optimal alignment between two (amino acid) sequences. The basic idea is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. We adopt this approach to find an optimal alignment between two traces. Let $\mathbf{t}_1$ and $\mathbf{t}_2$ be two traces. A matrix F indexed by i and j, is constructed where the value $F(i,j)$ is the score of the best alignment between the prefix $\mathbf{t}_1^i$ of $\mathbf{t}_1$ and the prefix $\mathbf{t}_2^j$ of $\mathbf{t}_2$. $F(i,j)$ is constructed recursively by initializing $F(0,0) = 0$ and then proceeding to fill the matrix from top left to bottom right. It is possible to calculate $F(i,j)$ if $F(i-1,j-1)$, $F(i-1,j)$ and $F(i,j-1)$ are known. There are three possible ways that the best score $F(i,j)$ of an alignment up to $\mathbf{t}_1^i$ and $\mathbf{t}_2^j$ could be obtained:

- $\mathbf{t}_1(i)$ could be aligned to $\mathbf{t}_2(j)$, in which case $F(i,j) = F(i-1,j-1) + \mathcal{S}(\mathbf{t}_1(i),\mathbf{t}_2(j))$; or
- $\mathbf{t}_1(i)$ is aligned to a gap, in which case $F(i,j) = F(i-1,j) + \mathcal{I}_R(\mathbf{t}_1(i-1),\mathbf{t}_1(i))$; or
- $\mathbf{t}_2(j)$ is aligned to a gap, in which case $F(i,j) = F(i,j-1) + \mathcal{I}_R(\mathbf{t}_2(j-1),\mathbf{t}_2(j))$.

The best score up to (i,j) will be the largest of these three options. In other words, we have

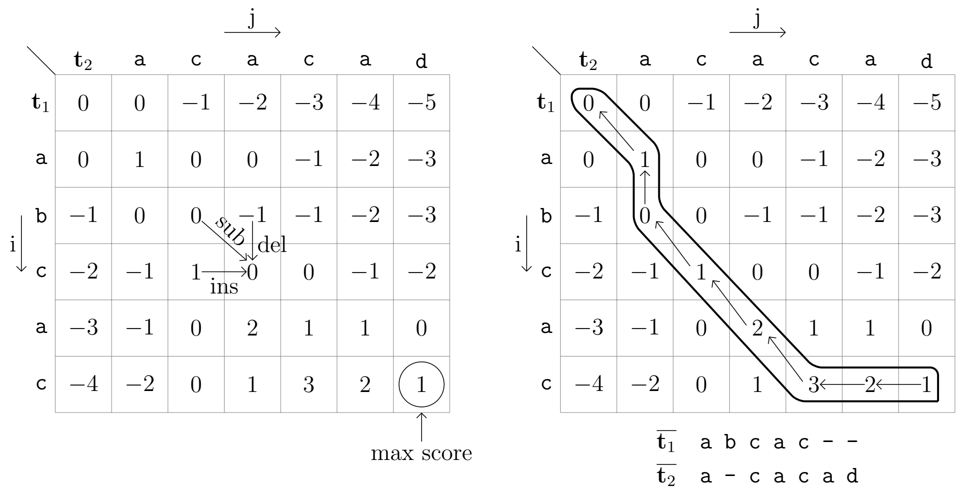
$$F(i,j) = \max \begin{cases} F(i-1,j-1) + \mathcal{S}(\mathbf{t}_1(i),\mathbf{t}_2(j)), \\ F(i-1,j) + \mathcal{I}_R(\mathbf{t}_1(i-1),\mathbf{t}_1(i)), \\ F(i,j-1) + \mathcal{I}_R(\mathbf{t}_2(j-1),\mathbf{t}_2(j)). \end{cases} \tag{8.1}$$

The values along the top row (when $i = 0$) and left column (when $j = 0$) need to be handled as follows. The values $F(i,0)$ represent alignments of a prefix of $\mathbf{t}_1$ to all gaps in $\mathbf{t}_2$. So, we can define $F(1,0) = 0$ and for $i > 1, F(i,0) = F(i-1,0) + \mathcal{I}_R(\mathbf{t}_1(i-1),\mathbf{t}_1(i))$.

Similarly, we can define $F(0, j)$. The value in the bottom right cell of the matrix, $F(|\mathbf{t}_1|, |\mathbf{t}_2|)$, is the best score for an alignment of $\mathbf{t}_1$ and $\mathbf{t}_2$.

To find the alignment itself, we must find a path of choices from Equation (8.1) that led to this best score, i.e., we move from the current cell (i, j) to one of the cells $(i-1, j-1)$, $(i-1, j)$, or $(i, j-1)$ from which the value $F(i, j)$ was derived. While doing so, we add a pair of symbols onto the front of the alignment: $\mathbf{t}_1(i)$ and $\mathbf{t}_2(j)$ if the step was to $(i-1, j-1)$, $\mathbf{t}_1(i)$ and the gap symbol '–' if the step was to $(i-1, j)$, or '–' and $\mathbf{t}_2(j)$ if the step was to $(i, j-1)$. At the end we will reach the start of the matrix, $i = j = 0$. The above procedure, called *traceback*, will retrieve only one of the alignments that gives the best score; there can be cases where multiple options of Equation (8.1) are equal. In these cases, an arbitrary choice is made. The set of all possible alignments for the best score can be enumerated by using graph traversal techniques.

Consider the two traces $\mathbf{t}_1$ = `abcac` and $\mathbf{t}_2$ = `acacad` and the unit score function. Figure 8.3(a) depicts the F matrix for these two traces using the unit score function. Consider the cell at row 4 and column 4, $F(3, 3)$; the value for this cell corresponds to the maximum of the score at $F(2, 2) + \mathcal{S}(\mathtt{c}, \mathtt{a})$ or $F(2, 3) + \mathcal{I}_R(\mathtt{b}, \mathtt{c})$ or $F(3, 2) + \mathcal{I}_R(\mathtt{c}, \mathtt{a})$. Since in the unit score function, the score of substitution of unlike activities and the score for indels is -1, we get $F(3, 3) = \max\{-1, -2, 0\} = 0$. The best score of the alignment is $F(5, 6) = 1$. Figure 8.3(b) depicts the traceback procedure pertaining to the best alignment. We start with the bottom right cell at $(5, 6)$ and identify the cells that led to the best score recursively until we reach the top left cell at (0,0).



$\mathbf{t}_1$ \ $\mathbf{t}_2$		a	c	a	c	a	d
	0	0	−1	−2	−3	−4	−5
a	0	1	0	0	−1	−2	−3
b	−1	0	0	−1	−1	−2	−3
c	−2	−1	1	0	0	−1	−2
a	−3	−1	0	2	1	1	0
c	−4	−2	0	1	3	2	1

```
t̄1  a b c a c - -
t̄2  a - c a c a d
```

(a) F-matrix (b) traceback

Figure 8.3: The F-matrix and the traceback computing an alignment between two traces using the unit score function.

8.2.3 Multiple Trace Alignment

We now consider the alignment of a bag of traces where the number of traces to align is more than 2. One of the best performing scoring mechanisms for multiple sequence alignment of genomic sequences is the *sum-of-pairs* (SP) method [10, 32]. Therefore, we adopt the sum-of-pairs method for trace alignment. Let $\overline{\mathbf{t}_h}$ and $\overline{\mathbf{t}_k}$ be two distinct rows extracted from a multiple trace alignment A over a bag of n traces (recall that trace alignment can be represented as a matrix A), and let $\text{Score}(\overline{\mathbf{t}_h}, \overline{\mathbf{t}_k})$ be the alignment score calculated in the same way as ordinary pairwise alignment of $\mathbf{t}_h$ and $\mathbf{t}_k$, then the SP score of a multiple trace alignment A is defined as

$$\text{Score}_{SP}(A) = \sum_{1 \le h < k \le n} \text{Score}(\overline{\mathbf{t}_h}, \overline{\mathbf{t}_k})$$

It is possible to generalize the pairwise dynamic programming alignment approach to the alignment of n traces. However, it is impractical for more than a few traces. Assuming that the traces are all of roughly the same length l, the space complexity of the multi-dimensional dynamic programming algorithm is $\mathcal{O}(l^n)$ and the time complexity is $\mathcal{O}(2^n l^n)$ [61]. Multiple sequence alignment that maximizes the SP score was shown to be NP-complete [258]. Since the computation of optimal multiple sequence alignment is prohibitively expensive, various heuristic algorithms have been proposed in literature [163].

We adopt the most popular heuristic approach [63], viz., the progressive alignment approach [67, 68], for trace alignment. The basic idea of progressive alignment is to iteratively construct a succession of pairwise alignments. Alignment is allowed between a pair of traces, a trace and an alignment, and between alignments. The selection of traces for alignment at each iteration is based on their similarity. Traces that are most similar to each other are aligned first. Once similar traces have been aligned, align the resulting clusters of traces against each other. A guide tree is built to assist this process. We use the Agglomerative Hierarchical Clustering algorithm (AHC) [113] (cf. Section 4.5) for generating this tree. The choice of AHC is due to the fact that it produces the tree naturally as a dendrogram while the tree has to be constructed subsequently if other clustering algorithms such as k-means are used.

Figure 8.4 illustrates an example of the progressive alignment strategy. In this example, we consider 5 traces. A guide tree is generated using AHC. Based on the guide tree, the traces $\mathbf{t}_2$ and $\mathbf{t}_3$ are aligned first using pairwise trace alignment. Traces $\mathbf{t}_4$ and $\mathbf{t}_5$ are aligned next using pairwise trace alignment. Subsequently, trace $\mathbf{t}_1$ is aligned with the alignment obtained from $\mathbf{t}_2$ and $\mathbf{t}_3$. Finally the two alignments obtained from the bag of traces $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$ and $[\mathbf{t}_4, \mathbf{t}_5]$ are aligned. While aligning an alignment A, with another alignment B, Equation (8.1) is modified as

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + \overline{\mathcal{S}}(C_A^i, C_B^j), \\ F(i-1, j) + \overline{\mathcal{I}_R}(C_A^{i-1}, C_A^i), \\ F(i, j-1) + \overline{\mathcal{I}_R}(C_B^{j-1}, C_B^j). \end{cases} \tag{8.2}$$

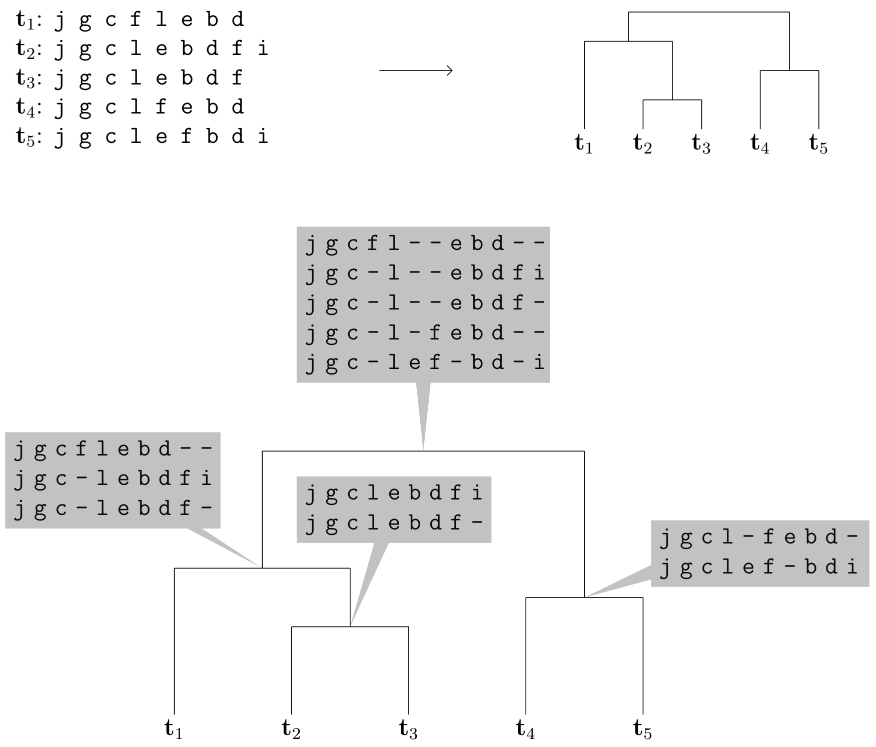


Figure 8.4: An example of progressive alignment approach for multiple trace alignment.

where $\overline{\mathcal{S}}(C_A^i, C_B^j)$ denotes the score of substituting column i of alignment A with column j of alignment B and is defined as

$$\overline{\mathcal{S}}(C_A^i, C_B^j) = \sum_{\mathtt{a},\mathtt{b} \in \mathcal{A}} n_A^i(\mathtt{a}) \cdot n_B^j(\mathtt{b}) \cdot \mathcal{S}(\mathtt{a}, \mathtt{b}) \tag{8.3}$$

where $n_X^i(\mathtt{a})$ denotes the frequency (count) of activity $\mathtt{a}$ in column i of alignment X. $\overline{\mathcal{I}_R}(C_A^{i-1}, C_A^i)$ denotes the score of inserting column i in alignment A given that its left column is $i-1$ and is defined as

$$\overline{\mathcal{I}_R}(C_A^{i-1}, C_A^i) = \sum_{\mathtt{a},\mathtt{b} \in \mathcal{A}} f_A^i(\mathtt{a}, \mathtt{b}) \cdot \mathcal{I}_R(\mathtt{a}, \mathtt{b}) \tag{8.4}$$

where $f_A^i(\mathtt{a}, \mathtt{b})$ is the frequency of activity $\mathtt{b}$ in column i of alignment A given that its neighboring activity is $\mathtt{a}$ in column $i-1$. The procedure for finding the "best" alignment is similar to that of pairwise alignment.

Note that the guide tree enables the visualization of alignments for different subsets of the traces. The alignment at the root of the tree corresponds to the alignment of all the traces in the event log whereas an alignment at any internal node of the guide tree depicts the alignment corresponding to the traces constituting the leaves of the sub-tree at the node. It is often the case that event logs contain traces capturing different execution behavior of a process and clustering assists in grouping together coherent sets of traces.

8.3 Framework

We propose the framework depicted in Figure 8.5 for trace alignment. The framework identifies the following steps:

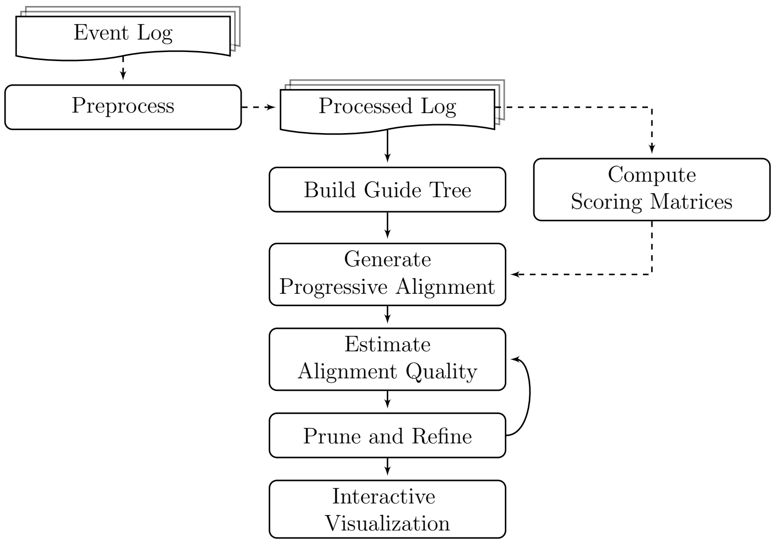


Figure 8.5: Framework for multiple trace alignment.

- *Preprocess:* Preprocessing involves steps such as removal of outliers, removal of loop-constructs, abstraction of activities, and transformation of log, etc. The detection and removal of outliers (explained later in this section) is critical for obtaining interesting alignments.
- *Compute Scoring Matrices:* As discussed in Section 8.2, alignments are sensitive to the substitution and indel score functions, $\mathcal{S}$ and $\mathcal{I}_R$ respectively. We use the approach presented in Section 4.4 for deriving the substitution and indel scores from the event log.
- *Build Guide Tree:* A guide tree assists in progressive alignment of multiple traces as illustrated in Figure 8.4. We use the Agglomerative Hierarchical Clustering (AHC) approach for building the guide tree. However, other approaches such as neighbor joining [203] can be used.
- *Generate Progressive Alignment:* The progressive alignment approach is used to compute the multiple trace alignment. The guide tree generated in the above step directs the growth of progressive alignment as a series of pairwise alignments.

- *Estimate Alignment Quality:* Progressive alignment is a heuristic approach. Therefore, the alignment that is obtained does not need to be optimal. Furthermore, any error in alignment done in early stages of progressive alignment cannot be undone (cf. advanced alignment techniques in Section 8.7). Hence it is essential to estimate the quality of an alignment. We use the *information score* as a measure of alignment quality. The information score of a column in an alignment is defined as $1 - E/E_{max}$, where E is the entropy of activities in the column. The entropy of a column is defined as $E = \sum_{\mathsf{a} \in \mathcal{A} \cup \{-\}} -p_{\mathsf{a}} \log_2(p_{\mathsf{a}})$ where p_{a} is the probability of occurrence of a in the column. E_{max} is the maximum entropy which is equal to $\log_2(|\mathcal{A}| + 1)$. We discuss on an advanced metric, viz., *misalignment score*, later in Section 8.4.

- *Prune and Refine:* Construction of multiple trace alignment is a very complex problem, and most heuristic algorithms usually fail to generate an optimal alignment. Disturbances in an alignment can creep in from many sources thereby making the final alignment far from optimal. Disturbances here refer to the misplacement of gaps in an alignment. Efficient techniques for pruning and refining alignments need to be supported. We will discuss more about this in the next section.

- *Interactive Visualization:* Apart from just pictorially depicting the alignment, it is desirable to have additional interactive features for the analysts to explore into the patterns and the alignments uncovered. Features such as editing an alignment, sorting and/or filtering alignment columns based on activities of interest would all lead to gaining further insights into the execution of processes.

Although the definition of what constitutes an outlier is left open, we adopt one simple definition of outliers based on the length of the traces. It could be the case that in an event log there are certain process instances whose lengths deviate a lot from the average trace length in the log, e.g., an event log has an average trace length of 50 activities (say, across 100 traces) while there are 5 traces with lengths above 250. Since an alignment is at least as long as the maximum trace length, such outlier traces in the log can lead to an alignment with too many gap symbols. Hence the removal of such traces is important. Note that the definition of outliers can change based on the perspective of analysis. If we are interested in finding common execution patterns or the backbone sequence of a process, the above definition of outliers may work fine. However, if we are interested in finding non-conforming traces or deviations in anomalous traces from normal traces, then the above definition might be inappropriate. Trace clustering as discussed in Chapter 4 can assist in removing such outliers. In addition, recent efforts in detecting and dealing with outliers in process mining such as [72, 80] can be adopted. The efficacy of these techniques in the context of trace alignment needs to be explored and is beyond the scope of this thesis.

The presence of loop constructs and the multiple invocations of a process fragment (subprocess) can all lead to variations in the lengths of the traces. Loop constructs and multiple invocations of process fragments manifest as tandem arrays and maximal repeats in the event logs respectively and can be detected efficiently in linear time as discussed in Chapter 3. One can identify such patterns, define abstrac-

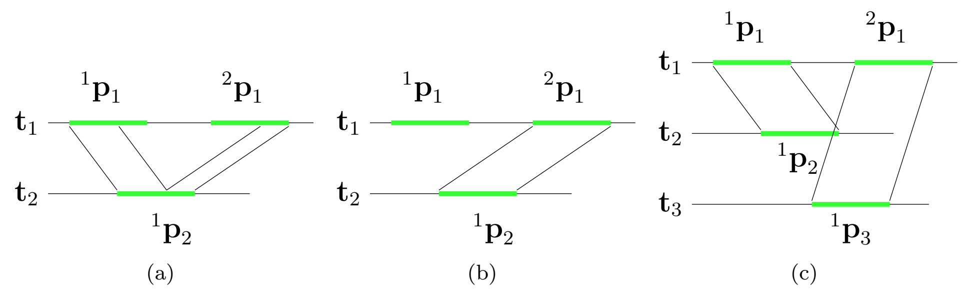


Figure 8.6: Scenarios of misalignment in the presence of recurring patterns of execution.

tions over them, and transform the log to a higher-level as discussed in Chapter 6. Trace alignment can then be applied on the transformed log.

8.4 Refining Alignments

Variation in the lengths of the traces, the choice of scoring functions used, the method and parameter choices used in the generation of a guide tree, and strategies used in resolving conflicts during traceback can all lead to disturbances or misalignments. Furthermore, misalignments in earlier stages of progressive alignment percolate to later stages. It is conjectured that detecting such misalignments and refining them might improve the quality of the final alignment. Misalignments are more pronounced in cases where there are recurring patterns of common execution behavior such as the manifestation of loops and multiple invocations of a functionality.

Figure 8.6 depicts three scenarios resulting in a misalignment. Let $\mathbf{p}$ be a sequence pattern, i.e., a sequence of activities. $^k\mathbf{p}_i$ indicates the k^{th} instance of the manifestation of $\mathbf{p}$ in the trace $\mathbf{t}_i$. In Figure 8.6, there are two instances of $\mathbf{p}$ in trace $\mathbf{t}_1$ and one instance of $\mathbf{p}$ in traces $\mathbf{t}_2$ and $\mathbf{t}_3$. Figure 8.6(a) depicts the scenario where the lone instance of $\mathbf{p}$ in trace $\mathbf{t}_2$ is split up and aligned to the two instances of $\mathbf{p}$ in $\mathbf{t}_1$. Figure 8.6(b) depicts the scenario where the lone instance of $\mathbf{p}$ in $\mathbf{t}_2$ is aligned to the second instance of $\mathbf{p}$ ($^2\mathbf{p}_1$) in $\mathbf{t}_1$ whereas one would have preferred it to be aligned with $^1\mathbf{p}_1$ (assuming that the contexts defined by events preceding and succeeding $^1\mathbf{p}_2$ is similar to that of $^1\mathbf{p}_1$). Figure 8.6(c) depicts an even more undesirable situation where the lone instance of $\mathbf{p}$ in $\mathbf{t}_2$ is aligned to the first instance of $\mathbf{p}$ in $\mathbf{t}_1$ and the lone instance of $\mathbf{p}$ in $\mathbf{t}_3$ gets aligned to the second instance of $\mathbf{p}$ in $\mathbf{t}_1$. What is worse is that although $\mathbf{p}$ manifests in both $\mathbf{t}_2$ and $\mathbf{t}_3$, they do not get aligned due to their alignment conflict with $\mathbf{t}_1$. In this example, a desirable alignment would be the one where the first instance of $\mathbf{p}$ in all the three traces are aligned together.

The above examples considered the scenario where the pattern $\mathbf{p}$ manifests exactly as is (alike) in all the traces. However, the presence of concurrent activities and/or optional activities in a process might disturb the manifestation of $\mathbf{p}$ in a trace. In other words, there could be instances where the concurrent/optional activities are interspersed in the manifestation of $\mathbf{p}$. For example, in the trace $\mathbf{t}$ = `jgclfebdklebdklebdi`, the concurrent activity `f` is interspersed in the manifestation of the first instance of the pattern `lebd`. Figure 8.7 depicts two scenarios of misalignments in the presence of concurrent/optional activities. In Figure 8.7(a), there exists a sequence of optional activities interleaved in ${}^1\mathbf{p}_2$. It could be the case that the alignment of $\mathbf{t}_1$ and $\mathbf{t}_2$ results in the prefix of ${}^1\mathbf{p}_2$ before the optional activities to be aligned with the prefix of ${}^1\mathbf{p}_1$. The optional activities and the suffix of ${}^1\mathbf{p}_2$ following the optional activities are aligned with gaps as indicated in Figure 8.7(a), resulting in an undesirable alignment. Instead, an ideal alignment would be the one that is depicted under 'preferred alignment' in this case. Figure 8.7(b) illustrates the existence of a sequence of concurrent activities interspersed in ${}^1\mathbf{p}_2$. A possible misalignment and the preferred alignment are also depicted in Figure 8.7(b). For simplicity, we illustrate the misalignment scenarios with both the traces having a single instance of $\mathbf{p}$ in Figure 8.7. However, one can imagine scenarios where multiple instances of $\mathbf{p}$ are manifested with uneven numbers across traces, and with some instances having an interleaved manifestation of optional/concurrent activities. In such cases, the misalignment scenarios depicted in both Figures 8.6 and 8.7 are (together) possible.

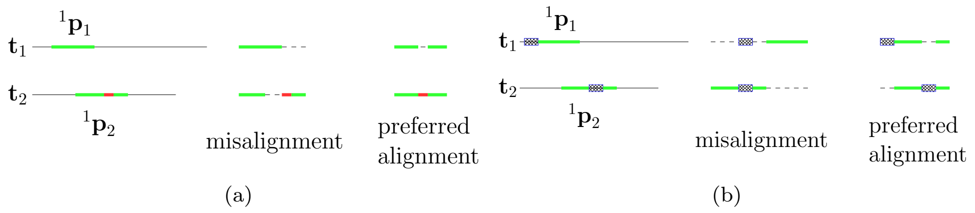


Figure 8.7: Scenarios of misalignment in the presence of concurrent/optional activities interspersed with recurring patterns of execution.

The information score metric defined earlier is not rich enough to capture such misalignments. Robust metrics to assess the quality of (mis-)alignment are needed. The definition of such quality metrics remains an open research topic in this area. We define a pattern based misalignment metric, viz., *misalignment score*, to capture the degree of misalignment with respect to the pattern.

Definition 8.2 (Misalignment Score). Let $\mathcal{L} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n]$ be a bag of traces and $\overline{\mathcal{L}} = [\overline{\mathbf{t}_1}, \overline{\mathbf{t}_2}, \ldots, \overline{\mathbf{t}_n}]$ be the corresponding bag of aligned traces of $\mathcal{L}$. Given a pattern $\mathbf{p}$, let $c(\mathbf{p}, \overline{\mathbf{t}_i})$ denote the frequency of occurrence (considering both the exact and interspersed/non-continuous manifestation, cf. Definitions 6.2 and 6.3) of the pattern $\mathbf{p}$ in its corresponding original trace $\mathbf{t}_i \in \mathcal{L}$. For any two aligned traces $\overline{\mathbf{t}_i}$ and $\overline{\mathbf{t}_j} \in \overline{\mathcal{L}}$, let $\overline{\mathbf{t}_u} = \arg\min_{\overline{\mathbf{t}} \in \{\overline{\mathbf{t}_i}, \overline{\mathbf{t}_j}\}} c(\mathbf{p}, \overline{\mathbf{t}})$ and $\overline{\mathbf{t}_v} = \arg\max_{\overline{\mathbf{t}} \in \{\overline{\mathbf{t}_i}, \overline{\mathbf{t}_j}\}} c(\mathbf{p}, \overline{\mathbf{t}})$. In other words, among $\overline{\mathbf{t}_i}$ and $\overline{\mathbf{t}_j}$, $\overline{\mathbf{t}_u}$ represents the one that has the lesser number of instances of the pattern $\mathbf{p}$ while $\overline{\mathbf{t}_v}$ represents the one with a higher number of instances of $\mathbf{p}$.

Let $\text{IN}(\mathbf{p}, k, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v})$ denote the set of all instance numbers of pattern $\mathbf{p}$ in $\overline{\mathbf{t}_v}$ to which some activity in the k^{th} instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is aligned to. If the k^{th} instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is not aligned to any instance of $\mathbf{p}$ in $\overline{\mathbf{t}_v}$, i.e., it is aligned to only gaps, then $\text{IN}(\mathbf{p}, k, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{\}$. The misalignment score matrix $\mathcal{MS} = [\text{ms}_{ij}]$ $(1 \le i, j \le n)$ of a pattern $\mathbf{p}$ over the bag of aligned traces $\overline{\mathcal{L}}$ is defined as

$$\text{ms}_{ij} = \text{ms}_{ji} = \sum_{k=1}^{c(\mathbf{p},\overline{\mathbf{t}_u})} \sum_{r \,\in\, \text{IN}(\mathbf{p},k,\overline{\mathbf{t}_u},\overline{\mathbf{t}_v})} |r-k| + \delta(k)$$

where $\delta(k) = 1$ if any activity in the k^{th} instance of the pattern $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is aligned to a gap or any other activity not in $\mathbf{p}$ in $\overline{\mathbf{t}_v}$; $\delta(k) = 0$, otherwise. $\overline{\mathbf{t}_u}$ and $\overline{\mathbf{t}_v}$ correspond to that aligned trace with the lesser and higher number of instances of $\mathbf{p}$ respectively among $\overline{\mathbf{t}_i}$ and $\overline{\mathbf{t}_j}$ as defined above. The $|r-k|$ signifies the distance of misalignment. ms_{ij} indicates the degree of misalignment pertaining to all instances of pattern $\mathbf{p}$ in $\overline{\mathbf{t}_u}$. ⌟

Let us look at the misalignment score metric with an example. Consider the two aligned traces

$\overline{\mathbf{t}_1}$ = `jgcl----f---ebdklebdi`

$\overline{\mathbf{t}_2}$ = `jgc-fleb-dklebdklebdi`

and the pattern $\mathbf{p}$ = `lebd`. Activity `f` is a concurrent activity in the process[1] and is interspersed in $\mathbf{p}$ in $\overline{\mathbf{t}_1}$. $c(\mathbf{p}, \overline{\mathbf{t}_1}) = 2$ and $c(\mathbf{p}, \overline{\mathbf{t}_2}) = 3$, i.e., there are two instances of $\mathbf{p}$ in $\mathbf{t}_1$ and three instances of $\mathbf{p}$ in $\mathbf{t}_2$. $\overline{\mathbf{t}_u} = \overline{\mathbf{t}_1}$ and $\overline{\mathbf{t}_v} = \overline{\mathbf{t}_2}$. $\text{IN}(\mathbf{p}, 1, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{2\}$ and $\text{IN}(\mathbf{p}, 2, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{3\}$, i.e., the first instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is partially aligned to the second instance of $\mathbf{p}$ in $\overline{\mathbf{t}_v}$ and the second instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is completely aligned to the third instance of $\mathbf{p}$ in $\overline{\mathbf{t}_v}$. $\delta(1) = 1$ since the activity `l` in the first instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is aligned to a gap while $\delta(2) = 0$. $\text{ms}_{12} = |2-1| + \delta(1) + |3-2| + \delta(2) = 3$.

As another example, let us consider the pattern $\mathbf{p}$ = `lebd` and the two aligned traces

$\overline{\mathbf{t}_3}$ = `jgclfebdklebdklebdklebdi`

$\overline{\mathbf{t}_4}$ = `jgclf-----ebdklebdklebdi`

$c(\mathbf{p}, \overline{\mathbf{t}_3}) = 4$ and $c(\mathbf{p}, \overline{\mathbf{t}_4}) = 3$, i.e., there are four instances of $\mathbf{p}$ in $\mathbf{t}_3$ and three instances of $\mathbf{p}$ in $\mathbf{t}_4$. $\overline{\mathbf{t}_u} = \overline{\mathbf{t}_4}$ and $\overline{\mathbf{t}_v} = \overline{\mathbf{t}_3}$. $\text{IN}(\mathbf{p}, 1, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{1, 2\}$, $\text{IN}(\mathbf{p}, 2, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{3\}$ and $\text{IN}(\mathbf{p}, 3, \overline{\mathbf{t}_u}, \overline{\mathbf{t}_v}) = \{4\}$, i.e., the first instance of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ is partially aligned to both the first and second instances of $\mathbf{p}$ in $\overline{\mathbf{t}_v}$ while the second and third instances of $\mathbf{p}$ in $\overline{\mathbf{t}_u}$ are completely aligned to the third and fourth instances of $\mathbf{p}$ in $\overline{\mathbf{t}_v}$ respectively. $\delta(1) = \delta(2) = \delta(3) = 0$. $\text{ms}_{34} = |1-1| + |2-1| + \delta(1) + |3-2| + \delta(2) + |4-3| + \delta(3) = 3$.

Once the misalignment score matrix is computed for a given alignment, the *cumulative misalignment score* is then defined to be the sum of the elements in the

[1]concurrent activities manifest in different columns across mutually exclusive traces in an alignment.

upper/lower triangle of the matrix (since the matrix is symmetric). An objective of a refinement technique is then to minimize the cumulative misalignment score. In the following subsections, we propose a few techniques of refining alignments.

8.4.1 Global vs. Semi-global Trace Alignment

The alignment procedure described in Section 8.2 is also called as *global trace alignment*. Depending on the scoring functions, global trace alignment can sometimes penalize gaps at the beginning and/or end of the traces in the alignment. In order to allow gaps to be inserted at the beginning/end of any trace in an alignment, a variant of the global trace alignment called the *semi-global trace alignment* can be considered. Here the best score of the alignment is defined to be the one that is the maximum in the last row or last column of the F matrix defined in Section 8.2. Traceback procedure starts from that cell and proceeds until it stops at the first position it reaches in the top row or left column. Gaps can then be inserted in the appropriate trace in the positions subsequent to the maximum value cell in the last row/column and prior to the position it reached in the top row or left column. Figure 8.8 depicts the difference between global trace alignment and semi-global trace alignment of two traces aligned using the same scoring functions.

$\overline{\mathbf{t}_1}$ `j g c - a h b - - - - f d`
$\overline{\mathbf{t}_2}$ `j g c f a h b d k a h b d`

(a) Global trace alignment

$\overline{\mathbf{t}_1}$ `j g c - a h b f d - - - - -`
$\overline{\mathbf{t}_2}$ `j g c f a h b - d k a h b d`

(b) Semi-global trace alignment

Figure 8.8: An example of global trace alignment and semi-global trace alignment.

In this example, the two traces have a common execution pattern $\mathbf{p}$ = `ahbd`. $\mathbf{t}_1$ has one instance of `ahbd` while $\mathbf{t}_2$ has two instances. Global trace alignment leads to the scenario depicted in Figure 8.6(a) where the lone instance of `ahbd` in $\mathbf{t}_1$ is split up and the two splits are aligned to different instances of `ahbd` in $\mathbf{t}_2$. This problem is mitigated with the semi-global trace alignment as depicted in Figure 8.8(b). The misalignment score for the pattern $\mathbf{p}$ is $\mathrm{ms}_{12} = 1$ for the resulting alignment using the global trace alignment while it is 0 for the one obtained using semi-global trace alignment. We recommend to consider semi-global trace alignment (at any iteration of progressive alignment) in scenarios where the traces to be aligned differ vastly in their lengths (for example, due to the manifestation of loop constructs).

8.4.2 Block Shift Refinement

Consider the bag of traces $\mathcal{L}$ = [`jgcflebdklebdklebdi`, `jgcflebdklebdi`, `jgclfebd-klebdklebdi`, `jgclfebdklebdklebdklebdi`, `jgclfebdklebdi`, `jgclebfdklebdkleb-di`, `jgclebdfklebdklebdi`, `jgclebdklfebdklebdi`]. Figure 8.9(a) depicts an alignment of these 8 traces. We can see that there exists a common execution pattern `lebd` in $\mathcal{L}$ and that there are different instances of `lebd` among the traces. A concurrent activity `f` has an interleaved manifestation among the traces. The misalignment scenarios depicted in Figures 8.6 and 8.7 can be observed in Figure 8.9(a). For example, the first instance of `lebd` in trace $\mathbf{t}_5$ is split up and aligned with the first and third instances of `lebd` in $\mathbf{t}_4$. The first instance of `lebd` in trace $\mathbf{t}_2$ is aligned with the second instance of `lebd` in $\mathbf{t}_1$. Figure 8.10(a) depicts the misalignment score matrix of the alignment in Figure 8.9(a). The cumulative misalignment score is 53.

```
jgc-f----leb--dkl-ebdklebdi
jgc----------f--l-ebdklebdi
jgclf-----eb--dkl-ebdklebdi
jgclfebdkleb--dkl-ebdklebdi
jgcl---------f----ebdklebdi
jgcl------eb-fdkl-ebdklebdi
jgcl------ebdf-kl-ebdklebdi
jgcl------eb--dklfebdklebdi
```

(a)

```
jgc-f----lebd--kl-ebdklebdi
jgc-f----lebd--kl-ebd-----i
jgclfebdklebd--kl-ebd-----i
jgclfebdklebd--kl-ebdklebdi
jgclfebdklebd-------------i
jgcl-eb------fdkl-ebdklebdi
jgcl-ebd-----f-kl-ebdklebdi
jgcl-ebdkl---f----ebdklebdi
```

(b)

```
jgc-f----lebd--klebdklebdi
jgc-f----lebd--klebd-----i
jgclfebdklebd--klebd-----i
jgclfebdklebd--klebdklebdi
jgclfebdklebd------------i
jgcl-eb------fdklebdklebdi
jgcl-ebd-----f-klebdklebdi
jgcl-ebdkl---f---ebdklebdi
```

(c)

Figure 8.9: An example of block shift refinement: (a) original alignment, (b) block shifted alignment with a gap column, and (c) block shifted alignment after the removal of gap column in (b).

	1	2	3	4	5	6	7	8
1	0							
2	2	0						
3	1	2	0					
4	3	4	3	0				
5	3	1	2	4	0			
6	1	2	0	3	2	0		
7	1	2	1	4	2	1	0	
8	1	2	0	3	2	0	1	0

(a) original alignment

	1	2	3	4	5	6	7	8
1	0							
2	0	0						
3	3	2	0					
4	3	2	0	0				
5	2	2	0	0	0			
6	1	1	3	3	2	0		
7	1	1	2	2	1	1	0	
8	2	3	3	2	1	2	1	0

(b) block shifted refined alignment

Figure 8.10: The misalignment score matrices (only the lower triangle is depicted).

We can see that the alignment can be improved by a mere adjustment/shifting of gap blocks. For example, the activity f and the first instance of lebd in $\mathbf{t}_2$ can be shifted to the left so that f gets aligned to f of $\mathbf{t}_1$ and lebd gets aligned to the first instance of lebd in $\mathbf{t}_1$. Algorithm 8.1 presents the refinement of an alignment by shifting non-gap activities to the left. The basic idea is to consider each trace of the alignment from left to right shifting an activity at a column preceded by a block of gaps to the left most possible column in that trace where that activity manifests in any other trace of the alignment. Figure 8.9(b) depicts the intermediary alignment after Step 13 of Algorithm 8.1. The shifting of activities has resulted in an alignment where there exists a column with only the gap symbol. Such columns can be removed. Figure 8.9(c) depicts the refined alignment after block shifts. Figure 8.10(b) depicts the misalignment score matrix of the block shifted refined alignment. It can be noticed that the alignment shown in Figure 8.9(c) is better than the one shown in Figure 8.9(a) (one can see that the common execution pattern lebd is much well conserved after refinement). Accordingly, the cumulative misalignment score of the refined alignment is 46 which is less than that of the original alignment.

Algorithm 8.1 Block Shift Refinement

```
Require: An alignment, $A$
 1: Let $m$ be the length of the alignment
 2: Let $\mathcal{A}_j$ denote the set of activities in column $j$ of $A$.
 3: for all aligned traces $\overline{\mathbf{t}_i}$ in $A$ do
 4:     for $j = 1$ to $m$ do
 5:         if there exists a block of gaps of length $g$ $(g \geq 1)$ starting at $j$ in $\overline{\mathbf{t}_i}$ then
 6:             if there exists a $k$, such that $j \leq k < j + g$ and $A(i, j + g) \in \mathcal{A}_k$ then
 7:                 swap $A(i, k)$ and $A(i, j + g)$. set $j = k$.
 8:             else
 9:                 set $j = j + g - 1$.
10:             end if
11:         end if
12:     end for
13: end for
14: Remove any column from $A$ that contains only the gap symbol
```

8.4.3 Concurrency Pruning and Realignment

Concurrent activities manifest in mutually exclusive traces across different columns in an alignment[2]. The basis for this arises from the fact that concurrent activities can have different contexts of execution. In the alignment of Figure 8.9(a), activity f is concurrent as it manifests in columns 5, 14, and 18 and there exists no trace that has f in more than one column. Concurrent activities are one of the primary

[2]This holds true only in scenarios where the concurrent activity is not involved in a loop construct. For cases where concurrent activities are involved in a loop, one can consider a subset of consecutive columns in an alignment demarcating an instance of the loop and then consider the manifestation of activities in that subset.

sources for misalignments. One can improve the quality of an alignment by identifying the presence of concurrent activities and handling them in a special way. Algorithm 8.2 presents a procedure for pruning concurrent activities and refining an alignment[3]. This algorithm only deals with the specific case of manifestation of concurrent activities in an alignment as defined in Step 2. Dealing with the other scenario where a concurrent activity is aligned with another (potentially concurrent) activity is relatively complex and is beyond the scope of this thesis. Figure 8.11 depicts the manifestation of a concurrent activity a in three different alignments. In Figure 8.11(a), the concurrent activity a is aligned to either itself or a gap in all the columns of the alignment where it manifests whereas in Figure 8.11(b) it is aligned to another activity c in the first column and in Figure 8.11(c) it is aligned to another concurrent activity b. Algorithm 8.2 is applicable to concurrent activities manifested as in the scenario shown in Figure 8.11(a) and dealing with the scenarios shown in Figures 8.11(b) and 8.11(c) is beyond the scope of this thesis.

Algorithm 8.2 Concurrency Pruning and Realignment

```
Require: An alignment $A$
1: Let $\mathcal{C}$ be the set of all potentially concurrent activities in $A$. These are activities that manifest in different columns across mutually exclusive traces in the alignment $A$.
2: Let $\mathcal{C}_I \subseteq \mathcal{C}$ be the set of all concurrent activities that are aligned to either only itself or a gap in the alignment.
3: for all a $\in \mathcal{C}_I$ do
4:     Let $A'$ be the alignment obtained from $A$ after removing the columns in which a manifests
5:     Perform block shift refinement on $A'$. Let $A''$ be the refined alignment.
6:     Insert the concurrent activity a as columns at appropriate positions in the refined alignment. Let $A'''$ be the new alignment.
7:     Set $A = A'''$.
8: end for
```

```
a    –    –
a    –    –
– ... a ... –
–    –    a
–    –    a
```

(a)

```
a    –    –
a    –    –
– ... a ... –
c    –    a
–    –    a
```

(b)

```
a     b
a     b
a ... b
b     a
b     a
```

(c)

Figure 8.11: Scenarios of manifestation of concurrent activities in an alignment. Here, we only consider (a).

Figure 8.12(a) depicts the alignment obtained from Figure 8.9(a) after removing the columns where the concurrent activity f manifests (Step 4, Algorithm 8.2).

[3]Pruning here refers to deleting the manifestation of the concurrent activity in the aligned traces.

```
jgc-----leb-dklebdklebdi    jgclebdklebdklebd-----i    jgcfl-eb-d-kl-ebdklebd-----i
jgc-----------lebdklebdi    jgclebdklebd----------i    jgcfl-eb-d-kl-ebd----------i
jgcl-----eb-dklebdklebdi    jgclebdklebdklebd-----i    jgc-lfeb-d-kl-ebdklebd-----i
jgclebdkleb-dklebdklebdi    jgclebdklebdklebdklebdi    jgc-lfeb-d-kl-ebdklebdklebdi
jgcl-----------ebdklebdi    jgclebdklebd----------i    jgc-lfeb-d-kl-ebd----------i
jgcl-----eb-dklebdklebdi    jgclebdklebdklebd-----i    jgc-l-ebfd-kl-ebdklebd-----i
jgcl-----ebd-klebdklebdi    jgclebdklebdklebd-----i    jgc-l-eb-dfkl-ebdklebd-----i
jgcl-----eb-dklebdklebdi    jgclebdklebdklebd-----i    jgc-l-eb-d-klfebdklebd-----i

           (a)                        (b)                          (c)
```

Figure 8.12: An example of concurrent activity pruning and realignment: (a) alignment obtained from Figure 8.9(a) after removing the columns where concurrent activity 'f' manifests, (b) alignment obtained after block shift refinement of (a), and (c) alignment obtained after inserting the concurrent activity in (b).

Figure 8.12(b) depicts the alignment obtained after block shift refinement on the alignment of Figure 8.12(a) (Step 5, Algorithm 8.2). It can be seen that this alignment is the ideal alignment of the traces without activity `f`. The next step corresponds to the reintroduction of the concurrent activity `f` into the alignment.

Let $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}_0$ be a function defined over the set of trace indices, $i \in \mathbb{N}$, and alignment column indices, $j \in \mathbb{N}$, that gives the activity index $k \in \mathbb{N}$ corresponding to the trace $\mathbf{t}_i$ at alignment column j provided $A(i,j) \in \mathcal{A}$ (if $A(i,j) = -$, then $f(i,j) = 0$). For example, for the alignment in Figure 8.9(a), $f(1,5) = 4$, because the activity in column 5 of the aligned trace $\overline{\mathbf{t}_1}$ is `f` and `f` is the fourth activity in $\mathbf{t}_1$. Similarly, $f(2,17) = 5$ and $f(3,19) = 11$. In the concurrency pruned and block shift refined alignment (A'' in Step 5, Algorithm 8.2), we insert a column at index j in the alignment with the concurrent activity in all the traces i (and gaps for others) if and only if the activity at $\mathbf{t}_i(f(i,j))$ corresponds to the concurrent activity. The resulting alignment after reintroducing the concurrent activity `f` in all the possible traces in the alignment of Figure 8.12(b) is as shown in Figure 8.12(c). In this refined alignment, the k^{th} instance of `lebd` (if it exists) among the traces are aligned with each other. *The misalignment score matrix for this refined alignment is a matrix of zero's (the cumulative misalignment score is zero)* which is an improvement from that of the original (53) and the block-shifted refined alignment (46). In this log, the patterns are only obscured by the presence of the concurrent activity `f` and an appropriate handling of the concurrent activity during refinement led to the conservation of the patterns in the alignment.

8.5 Computational Complexity

The major computation steps in the proposed approach pertain to the computation of guide tree and the progressive alignment of traces. As discussed earlier, we use the agglomerative hierarchical clustering algorithm to build the guide tree, which can be constructed in $\mathcal{O}(n^2)$ time for a given log containing n traces (we discussed the complexity of AHC in Section 4.6).

The time and space complexity of a single iteration of progressive alignment is $\mathcal{O}(l^2 + kl)$ [62] where l is the average length of the traces/profiles considered for alignment and k is the number of traces involved in the alignment in that iteration. There exists $n-1$ iterations in the progressive alignment thus making the alignment time to be polynomial. However, it is to be noted that during the initial iterations, the value of k is much smaller than the total number of traces. Furthermore, unless the traces to align are heterogeneous, the length of the aligned traces tends to be closer to the average trace length of the input traces.

8.6 Experiments and Discussion

We first illustrate the significance and effectiveness of trace alignment in process diagnostics and later expend on the scalability issues. We consider an event log containing cases pertaining to copy/scan job requests with at most 2 pages to copy/scan. This event log contains 21 distinct cases and 777 events distributed over 35 activities (a small log was chosen for legibility issues). The event log contains 4 outlier cases where some activities are reordered or skipped. The event log is first encoded into activity sequences (traces) where each activity is encoded with an alpha-numeric character. Figure 8.13 depicts the result of trace alignment. *Common execution patterns are captured in the form of well conserved regions.* For example the activity sequence `syiEHDr` corresponds to the Capture Image subprocess (`s`-Illuminate Document, `y`-Move Scan Head, `i`-Focus Light Beam, `E`-A/D Conversion, `H`-Interpolation, `D`-Filtered Image, `r`-Collect Image). The Capture Image subprocess contains a loop that is executed once for each page. Since the event log contains cases with at most 2 pages to copy/scan, the Capture Image process is executed once or twice.

Concurrent activities manifest in mutually exclusive traces across different columns in the alignment. It can be seen that the activities `l` (Y-Zoom), `e` (Load Quantizing Pixel), `w` (Calculate Quantization Error), and `h` (Rotate) are concurrent. The activity Y-Zoom is concurrent with another activity, X-Zoom (`a`), in the Image Processing subprocess. Similarly, the activities 'Load Quantization Pixel' and 'Calculate Quantization Error' are concurrent in the Half Toning subprocess. *Distinct activities that manifest in a single column share the same contexts and can potentially be activities modeled under a choice construct.* For example, the activities `b` (Send FTP) and `u` (Send SMTP) manifest in the same column in the alignment. They are indeed modeled using the XOR construct and hence are mutually exclusive.

Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled, i.e., regions with lots of gap symbol (–). Trace N8 has an exceptional activity `d` (Job Complete) in column 5. The trace also has a missing activity `s` (Illuminate Document) in column 6 (all the other traces have the activity

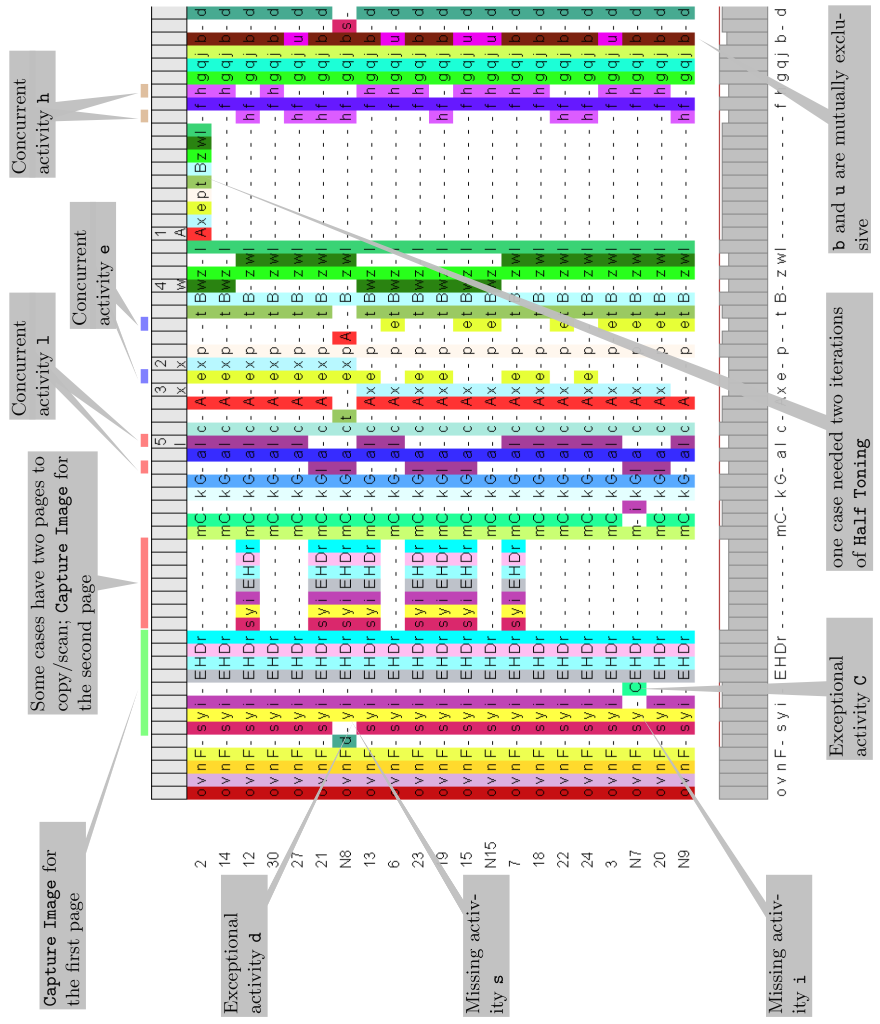

Capture Image for the first page
Some cases have two pages to copy/scan; Capture Image for the second page
Concurrent activity l
Concurrent activity e
Concurrent activity h
Exceptional activity d
Missing activity s
Missing activity i
Exceptional activity C
one case needed two iterations of Half Toning
b and u are mutually exclusive

Figure 8.13: Visualization of the aligned traces. Each row refers to a process instance. Columns describe positions in traces. Consider now the cell in row y and column x. If the cell contains an activity name **a**, then **a** occurred for case y at position x. If the cell contains no activity name (i.e., a gap "-"), then nothing happened for y at position x. Trace alignment aims a minimizing the number of gaps and maximizing the consensus.

s in column 6). The missed activity s appears in the *penultimate* column in N8. This is one of the outlier cases where the activities s and d are reordered. All the traces except N9 have activity x (Neighbor Quant Error Packing). Note that the omission of x in N9 is a violation of the expected behavior because this is a mandatory activity in the Half Toning subprocess. Another example: only the first trace has the activity sequence AxeptBzwl in the columns 44 to 52. This activity sequence corresponds to the Half Toning subprocess. This rare execution indicates that this case required two iterations of the Half Toning procedure as part of enhancing the image quality.

As just illustrated, trace alignment can assist in uncovering extremely interesting insights and can be used to probe process execution behavior. Trace alignment provides a complete perspective of activity executions in a log including that of long range dependencies (any dependencies between activities are reflected as common execution patterns in the traces where they manifest). Furthermore, with rich interactive visualization (such as the options for filtering/sorting columns containing an activity), trace alignment provides a flexible tool for the inspection of the log. Figure 8.14 summarizes how trace alignment assists in answering some of the diagnostic questions raised in this chapter. In the next two sections, we report on the scalability of the approach.

8.6.1 Influence of the Number of Traces

As discussed in Section 8.5, the computational complexity of trace alignment depends both on the length of the traces or profiles being aligned and the number of traces. In order to study the influence of the number of traces on trace alignment, we considered multiple event logs pertaining to the copy/scan job requests with varying number of traces but the same mean trace length. The length of a trace in copy/scan jobs is primarily affected by the Capture Image and Half Toning subprocesses (due to the loop constructs), the former depending on the number of pages to copy/scan and the latter depending on the image quality. We fixed the number of pages to scan to at most 2 and the number of iterations of Half Toning to at most 3. We simulated multiple event logs for varying number of cases with these characteristics. Figure 8.15(a) depicts the average time[4] along with the 95% confidence intervals (over 5 independent runs) to compute the alignment for varying number of traces (the mean trace length is 37 in these logs). Only the time taken to perform the alignment is depicted (the time to compute the guide tree is not considered as we have discussed this in detail in Chapter 4). Figure 8.15(a) also depicts the final alignment length. The alignment time is affected by the number of traces in each iteration of the progressive alignment; the number of traces being small during the initial iterations. In the worst case, we expect a quadratic complexity on the number of traces. However, we can observe a quasi-linear influence of the number of traces on the alignment time. This can be attributed to the shorter traces and less variability among them.

[4]All the computational times reported in this chapter are measured on an i3 Core CPU M350 @ 2.27 GHz with 4GB RAM running a 64-bit Windows 7 OS.

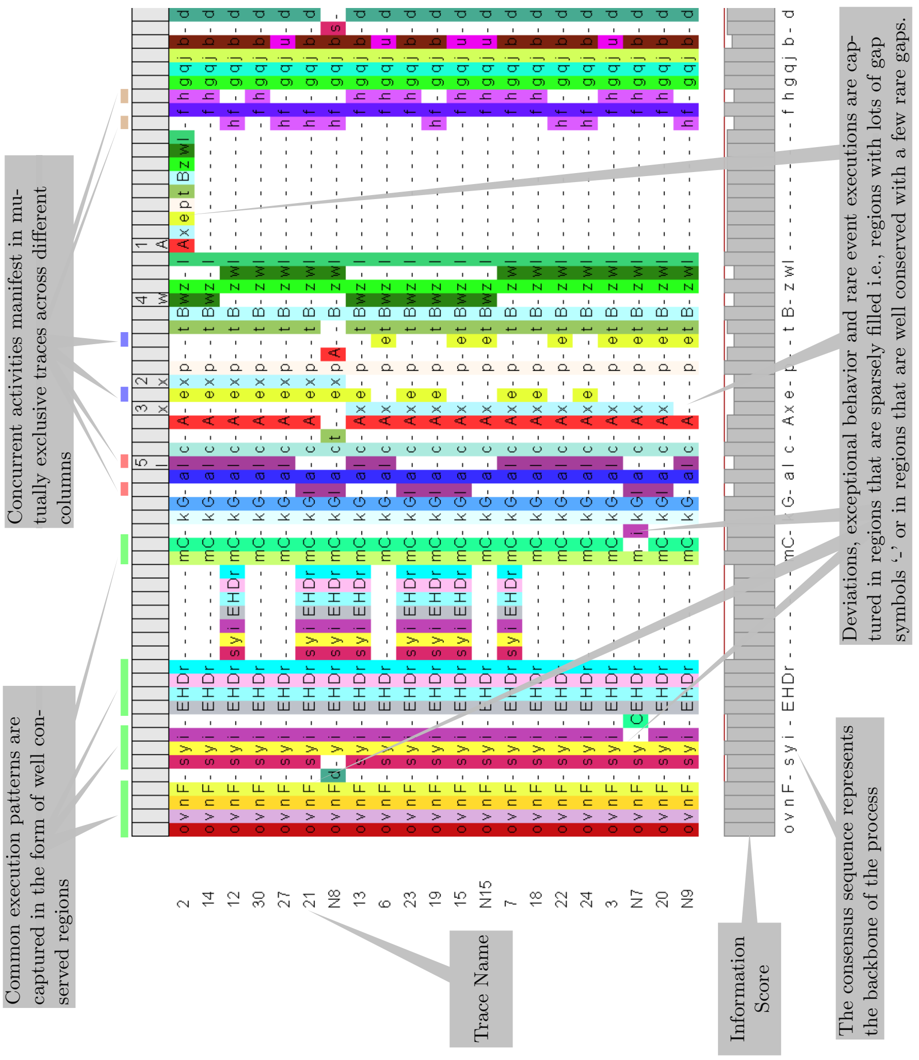
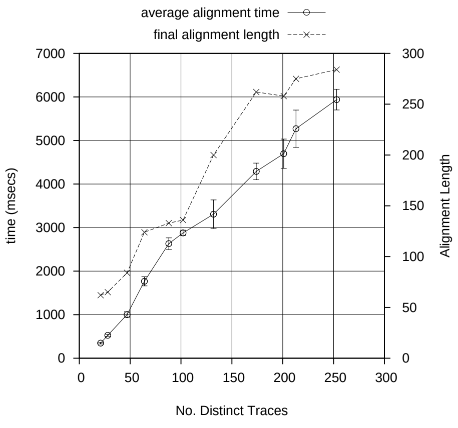
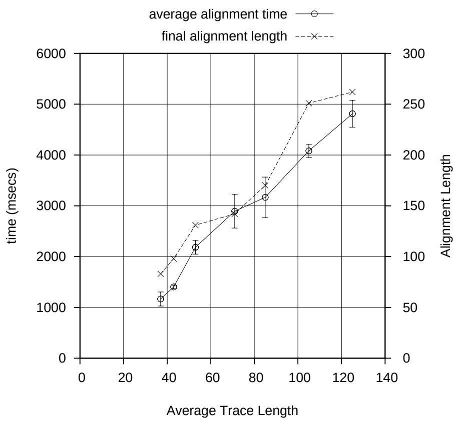

Concurrent activities manifest in mutually exclusive traces across different columns
Common execution patterns are captured in the form of well conserved regions
Trace Name
2
14
12
30
27
21
N8
13
6
23
19
15
N15
7
18
22
24
3
N7
20
N9
Information Score
The consensus sequence represents the backbone of the process
o v n F - s y i - E H D r - - - - - - - - m C - k G - a l c - A x e - p - - t B - z w l - - - - - - - - - - f h g q j b - d
Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lots of gap symbols '-' or in regions that are well conserved with a few rare gaps.

Figure 8.14: Visualization of the aligned traces along with annotations indicating how trace alignment can assist in answering some of the diagnostic questions.



(a) Influence of number of traces



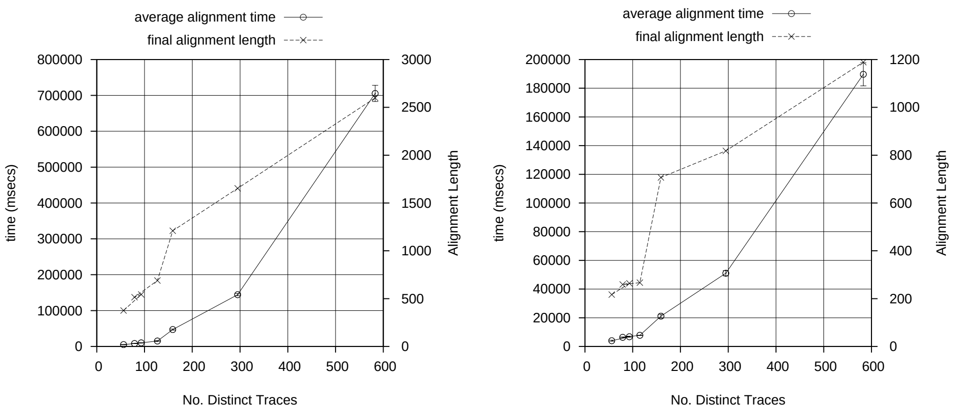
(b) Influence of trace length

Figure 8.15: Influence of the number of traces and trace length on the alignment time and alignment length.

8.6.2 Influence of Trace Length

In order to study the influence of the lengths of the traces on trace alignment, we set the number of traces to be the same. We vary the mean trace length by varying the number of pages to copy/scan. We simulated the digital copier process to generate multiple event logs, all containing 37 traces but with varying mean trace lengths. Figure 8.15(b) depicts the average time along with the 95% confidence intervals (over 5 independent runs) to compute the alignment for varying mean lengths of the traces. As discussed in Section 8.5, the length of the traces is expected to have a quadratic effect on the alignment computation time. However, we observe a more moderate increase in computation time in Figure 8.15(b). This can be attributed to the small number of traces (just 37) and less variability among them.

In a realistic scenario, as the size of the log changes, both the number of traces and the mean trace length change. We simulated multiple event logs to reflect a realistic scenario (here the number of pages to copy/scan can vary between 1 and 15 and the number of iterations in Half Toning can vary until the error reaches a certain threshold). We randomly added noise by reordering activities in the traces of this event log (around 18% of the traces are outliers). We used the MRA feature set along with the Euclidean distance metric to generate the guide tree. Figure 8.16(a) depicts the average time along with the 95% confidence intervals (over 5 independent runs) to compute the alignment for the different logs. Figure 8.16(a) also depicts the final alignment length. Unlike Figure 8.15, we now see the alignment time increasing quadratically. It is also to be seen that the final alignment length in these logs is relatively much larger when compared to Figure 8.15. This is an effect of the outlier traces present in the event log. As discussed in Chapter 4, the MRA feature set is able to segregate the outlier and valid traces into two clusters. The two clusters are



(a) Alignment including the root node (b) Alignment excluding the root node

Figure 8.16: Influence of varying log sizes on alignment time and alignment length.

the last ones to be merged in the AHC algorithm. Assume that we stop short of finding the alignment at the root node (i.e., we do not align the alignments obtained over the valid and outlier traces). Figure 8.16(b) depicts the alignment time in this scenario (we notice a quadratic behavior here as well). Figure 8.16(b) also depicts the final alignment length over the valid traces. We can see that the majority of time to compute the whole alignment is taken for the alignment at the root node (this is not surprising because at the root node, the number of traces as well as the length of the profiles to align is large). Furthermore, the final alignment length is almost doubled by aligning the alignments over the valid and outlier traces. *We recommend that in cases where the event log is heterogenous, the alignments be performed on smaller homogenous partitions of the event log rather than on the whole event log.*

8.7 Outlook

Finding high quality alignments is notoriously complex. In this section, we highlight some of the extensions that can be done to further improve the results of our trace alignment approach. We also mention some of the challenges related to trace alignment and directions for future research.

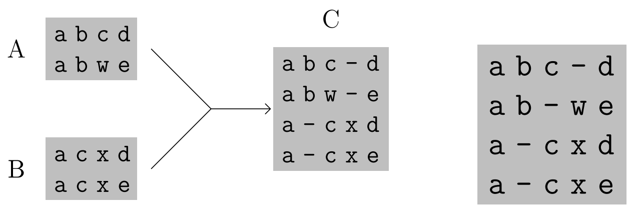
- *Multi-phase approach:* In order to deal with complex event logs (logs with a large number of activities/event classes, etc.), one can try to find alignments using a multi-phase approach. The basic idea is to first simplify the log by defining abstractions over the activities and then transforming the log with these abstractions as discussed in Chapter 6. Sub-logs are created for each abstract activity during this transformation. Trace alignment is then applied to the most abstract log with a provision to zoom into an abstract activity. Upon zooming into an abstract activity, an alignment obtained on the traces defined in the sub-log of the abstract activity is shown. The definition of abstractions and simplification of logs can be enabled by incorporating semantics in the log specification [50, 55] or through (semi-)automated means as discussed in Chapters 3 and 6.

- *Outlier/noise detection:* Noisy data poses a risk of misleading the alignment procedure and thereby resulting in low quality alignments. There is a need for techniques to identify difficult or even un-alignable bags of traces. For certain perspectives of analysis such as finding common execution patterns or backbone/critical elements of a process, such traces can be safely ignored. However, for certain other perspectives such as finding deviations/non-conforming traces, it is a tricky proposition to filter outliers as non-conforming traces could be treated as outliers. Techniques developed within the context of process mining such as [72, 80] for noise/outlier detection and filtering, and identifying coherency in event logs need to be explored. In addition, one can take leverage of the techniques in data mining and machine learning community and adapt them (if need be) by providing a process tinge to them.

- *Influence of guide tree:* As discussed in Section 8.3, progressive alignment suffers from the limitation that any error in alignment occurring in early stages of progressive alignment cannot be undone. Furthermore, the final alignment strongly depends on the order of traces being aligned. An improper choice of the traces to align in the initial stages might lead to misalignments which percolate to later stages. It is important to explore how different guide trees influence the quality of the alignment. Any ill-effects due to guide trees need to be mitigated.

- *Refinement strategies to improve the quality of an alignment:* In this chapter, we have presented three different refinement techniques to deal with misplaced gaps. However, robust quality metrics along with realignment strategies to effectively deal with the variations that might arise due to the manifestation of different process model constructs is highly desirable. Defining and identifying problematic regions in an alignment is a prerequisite to any refinement technique. Automated means of identifying such problematic regions is non-trivial and is in itself an interesting area of research.

- *Advanced alignment techniques:* The progressive alignment approach adopted for trace alignment is susceptible to converging to local optima. This is due to the fact that when aligning an individual trace or an alignment A with another alignment B, the traces in alignment A is not aligned directly and optimally with every internal trace within the alignment B but with the whole alignments A and B taken as atomic entities. Gaps in the resulting alignment are inserted as a column into the sub-alignments A and B as a whole, but are not optimally distributed throughout the resultant alignment. This is largely related to the influence of guide tree on final alignments. Figure 8.17 depicts an example of this phenomenon. For this example, let us assume that $\mathcal{S}(\mathtt{w},\mathtt{x}) > 1.5\mathcal{S}(\mathtt{w},\mathtt{c})$ (i.e., the substitution score of `w` with `x` is at least 1.5 times greater than the substitution score of `w` with `c`). The alignment of `w` with `c` in the alignment A in Figure 8.17(a) is frozen and cannot be altered in subsequent steps. When alignment B is aligned with alignment A, we obtain the alignment C as depicted in Figure 8.17(a). For the constraint on substitution scores as mentioned above, this is not an optimal alignment of the four traces. The optimal alignment for these traces is as depicted in Figure 8.17(b). Since $\mathcal{S}(\mathtt{w},\mathtt{x})$ is greater than $\mathcal{S}(\mathtt{w},\mathtt{c})$, an alignment technique should favor the alignment of `w` with `x`. However, in progressive alignment approach, the alignment of activities done in earlier stages cannot be altered in later stages.



(a) local optima (b) global optima

Figure 8.17: An example of progressive alignment approach converging to local optima. If the substitution score of `w` with `x` is at least 1.5 times greater than the substitution score of `w` with `c`, then the SP-score of (b) is greater than the SP-score of (a).

Also, the splitting up of common patterns of execution as discussed in Section 8.4 is an effect of such local optima. Recent advances in multiple sequence alignment (such as the one that takes into consideration some user defined constraints [155] while performing an alignment) need to be explored. One can specify the conservation of common execution patterns in the final alignment as a constraint. Such common execution patterns can be discovered automatically as discussed in Chapter 3. Other techniques such as the partial-order alignment [134] and the hybrid approach combining the progressive and partial-order alignments [85] have been shown to produce better alignments.

8.8 Conclusions

In this chapter, we proposed a novel approach of aligning traces and showed that this approach uncovers interesting patterns and assists in getting better insights into the actual behavior. We showed how trace alignment can help to provide new and valuable insights and in answering some of the interesting questions in process diagnostics raised in this thesis. We also showed that this approach is scalable for large event logs. Automatic generation of high-quality alignments is still challenging and there is much to be done to fully exploit the potential of this approach.

Chapter 9
Signature Discovery

In the previous chapter, we have looked at trace alignment as an approach for process diagnostics. We showed that trace alignment can be used to answer several of the diagnostic questions described in Section 1.4. In this chapter, we extend the work on process diagnostics by addressing a specific question: *can we find patterns in an event log that can discriminate between different classes of behavior?*. For example, consider the traces depicted in Figure 9.1. The traces carry a label indicating a particular behavior, ✔, ✖, or ✚. The figure also depicts the patterns that discriminate between ✚ and ✖ with ✔. More specifically, the three event patterns represented by circles are specific to traces that are labeled ✖ while the patterns represented by bars are specific to traces labeled as ✚. The presence of these patterns clearly distinguishes a trace from traces labeled ✔.

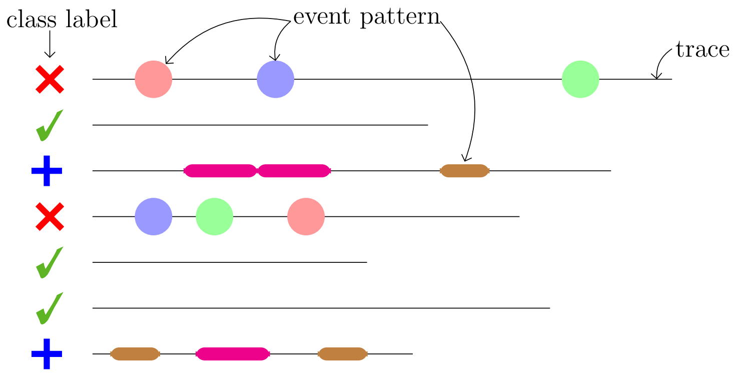


Figure 9.1: Essence of signature discovery. Given an event log where the cases (traces) are classified into different categories, the objective is to find patterns that discriminate between the different classes.

Signature discovery is concerned with finding patterns to discriminate between traces. Such patterns can be used to diagnose differences and predict the class of unclassified traces. There are many applications for signature discovery. We mention two motivating examples:

- *Fault diagnosis of high-tech systems:* High-tech systems such as medical devices, copier machines, and wafer scanners, all generate event logs capturing their day-to-day operations. These systems may malfunction when they are used abnormally (operational processes deviating significantly from their normal/intended usage). Malfunctions are also noticed when parts/components in the system encounter faults and/or deteriorate. System event logs are often the primary source of information for diagnosing (and predicting) the causes of failures in these systems. Early detection and diagnosis of system malfunctions can help avoid catastrophic failures and reduce productivity loss. For large and complex systems such as these, there is a pressing need for better techniques for processing, understanding, and analyzing these data [164]. We use the fault diagnosis of X-ray machines as a case study in this chapter.

- *Detecting fraudulent claims:* Insurance companies across all sectors (e.g., healthcare, automobile, property, etc.) are plagued by fraudulent claims costing billions of dollars annually [57]. Fraud often involves complex patterns of very minute indicators over a long period of time [192]. Detecting fraud and abuse relies heavily on analysts/auditors inspection of claims in conjunction with domain knowledge. There is a need for analytical techniques for effective detection of fraud [19, 115, 127, 271]. Assuming that there exists a historical database where we have "cases" comprising the evidence collected so far that indicates fraud, one can try to learn patterns/characteristics of behavior in such cases that discriminate them from normal behavior and use the uncovered patterns for monitoring future instances.

To answer the question raised above one may use trace alignment; align the traces and identify differences between traces of different classes. However, this requires manual inspection of the alignment to uncover the discriminating patterns. Inspecting for patterns is cumbersome for large datasets. Therefore, in this chapter, *we explore the feasibility of automatically extracting signature patterns from event logs, which can be associated to a particular class.* This assumes that the cases in an event log have an assigned class label (at first for training purposes). We propose a framework that incorporates well-known machine learning algorithms to derive the signatures. These algorithms require the data to be presented in a tabular form. Therefore, one of the steps in our framework is to translate the cases in an event log into class labeled feature vectors. The algorithms then learn the correlations among the features (called the training phase) and returns a classifier, which can be used over future instances.

The remainder of this chapter is organized as follows. Section 9.1 presents the related work on (signature) pattern discovery based on event correlations, more specifically, in the context of fault diagnosis. Section 9.2 introduces our case study, i.e., fault diagnosis of X-ray machines, in more detail. Section 9.3 presents the system event logs recorded by X-ray machines and discusses the data selection process for signature discovery. Section 9.4 presents our generic framework for signature discovery based on machine learning techniques. Section 9.5 discusses the results of applying the framework. Finally, Section 9.6 concludes the chapter.

9.1 Related Work

Any framework or methodology that attempts at discovering signature patterns, which discriminate between classes of behavior, is bound to use machine learning/-data mining techniques. The differences between the solutions mainly stem from the nature of application/domain, input data and its treatment, and the definition and scope of patterns. Our proposed framework for signature discovery is no different and is generic. It is based on two of the classical machine learning techniques, viz., decision trees [176, 177] and association rule mining [5, 138]. However, there are also two distinguishing aspects: firstly, the proposed framework allows for certain input data items to not have a class label, and secondly, the framework supports a wide range of features that are context-aware (we adopt the features defined in Chapter 4). Furthermore, the case study of fault diagnosis of X-ray machines makes it a unique proposition both from an application/domain point of view (cf. Section 9.2) and the nature of input data. The event logs from these systems need special treatment, which we discuss in Section 9.3. In the remainder of this section, we focus on related work in the context of fault diagnosis.

Event correlation based approaches for failure diagnosis have been proposed in [25, 148, 189]. There are also commercial tools such as HP's OpenView Self-Healing Services [109] and IBM's Trivoli [112] for network management. These methods and tools rely on either an existing rule base (typically derived from the Failure Mode and Effect Analysis (FMEA) [210]) or some known dependency models about the system. Either of these is hard to obtain for complex distributed systems and/or flexible systems such as medical systems. Automated identification of probable causes of performance problems in large server systems was proposed in [111]. This approach relies on the availability of well defined measurements on known metrics relevant to performance problems. Techniques such as these work well when one knows apriori what is to be measured; the analysis then focuses mainly on finding correlations over the measured values. However, event logs from high-tech systems such as X-ray machines capture all events that are trigged on/by/within the system and are typically designed for multiple applications (e.g., understanding system usage, debugging software bugs, etc.). These event logs tend to be fine-granular making the analysis challenging. Such fine-grained event logs first require elaborate preprocessing such as defining abstractions and selecting an appropriate scope for analysis, which can vary depending on the domain and application. We will later look at specific techniques of data selection and preprocessing for our case study on fault diagnosis of X-ray machines. Techniques based on the assumption that deviations exist in component interaction patterns during system/application failures have been proposed in [34, 135, 272]. However, these techniques cannot be applied to event logs that do not capture component interactions explicitly.

The problem of signature discovery can essentially be viewed as one aimed at inducing a classifier for an event log with labeled traces. Folino et al. [72] have proposed a decision tree based predictive model defined over a set of attributes. The approach that we present in this chapter also includes decision trees. However, our approach differs from [72] in three aspects: (i) in addition to decision trees, our approach also considers association rules between attributes and the class labels, (ii) our approach also addresses the scenario where only a subset of traces in the event log have a label, and (iii) we define several context-aware attributes over common execution patterns manifested in the traces. The approach presented in this chapter is an extension of the ideas presented in [139].

9.2 Fault Diagnosis of X-ray Machines

In this section, we present the case study of fault diagnosis of medical systems from a global leader in professional and consumer healthcare. Some of the organization's offerings include diagnostic imaging systems, healthcare IT, patient monitoring, cardiac devices, and customer services. An interventional X-ray (iXR) system is a large scale (consisting of thousands of parts), complex, and flexible imaging system designed to diagnose and treat *cardio* and *vascular* diseases. Although it is undesirable for these systems to malfunction, in reality, these systems do malfunction in their lifetime. However, when they do, it is important that these problems are quickly and predictably corrected. Good performance and robustness of these machines are important for a number of reasons: for clinicians to smoothly perform their interventional procedures, for hospitals to optimize healthcare costs, and above all it is critical for patients.

Whenever a system malfunctions, customers may contact the customer service department. Depending on the nature of the problem, the customer service personnel may try to resolve the issue remotely or may send a field service engineer (FSE) to the customer site. A typical workflow of a FSE during a corrective maintenance visit consists of the following steps: (a) diagnostics, (b) part(s) replacement, (c) part(s) configuration and calibration, (d) system verification, and (e) miscellaneous (e.g., ad-hoc support for other systems). Among the above steps, diagnostics stands out to be the most difficult and the most un-predictable step, under the assumption that all the steps are performed by trained personnel in accordance with established maintenance procedures. The effect of this is that

- for some of the problems, large deviations[1] in the Mean-Time-To-Repair (MTTR) are observed
- some of the problems could not be fixed correctly on the first time. This implies that *false* replacements are possible and/or additional visits could have been required to fix the problem

Furthermore, for *intermittent* problems, diagnostics pose an even greater challenge. The organization feels that efficient diagnostics and repair of problems could go a long way in improving customer satisfaction and also in enhancing its growth.

iXR systems installed across the globe continuously log all major events (e.g.,

[1]we do not provide the quantitative values due to confidentiality reasons

system operations, warnings, errors, etc.) during each day. Customer service personnel and field service engineers also make records of problems (customer complaints) and actions performed, as job sheets. Both of these data sources make up a set of historical service data. The organization sees an opportunity of improving their system maintenance through *log-based fault diagnosis*. More specifically, *they are interested in investigating whether the diagnostic value of system logs can be improved by discovering patterns that can be correlated to a known problem and/or corrective action with high confidence*. This turns out to be a non-trivial problem due to the following reasons:

- the system logs exhibit all of the characteristics mentioned in Section 1.2 (e.g., fine-granular, heterogenous, voluminous, etc.) thereby making the analysis challenging, analogous to searching for *needles in a haystack*, and
- iXR systems are large scale machines with many different things happening in a distributed manner and with complex relations between parts (e.g., the breakdown of one part could influence the behavior of other parts)

In other words, due to interactions among the system hardware and software components, the system event logs are comprised of streams of interleaved events, and only a small fraction of the events are relevant for the diagnosis of a given problem.

9.3 Data Selection and Preprocessing

In this section, we describe the iXR system logs and job sheets in more detail and discuss the data preparation process for signature discovery.

9.3.1 iXR System Logs and Job Sheets

Each iXR system records events during its daily operations. There are different types of events that are recorded, e.g., commands, errors, warnings, etc. Each entry in the system log contains information about a particular event and is characterized by a few attributes that provide rich information about the event. Some of these attributes include *event id* (the identifier of this event class), *unit* (the unit/component in the system that this event belongs to)[2], *severity* (indicating whether the event is of type error, warning, command, etc.), *description* (additional information pertaining to the event), *system mode* (indicating the mode of the system in which the event was trigged, e.g., start-up, shut-down, warm restart, normal, field service, etc.), and *timestamp*. It is important to note that iXR systems are constantly improved with newer versions of software (as updates). Such upgrades are meant to provide a better experience for the customers, e.g., functionality, features, performance, etc. At times, hardware upgrades also take place. The definition of event classes could change when transitioning to a newer version.

While the system logs capture the execution behavior of the system and the processes

[2]Being a large scale machine, the iXR system is divided into sub-systems, which themselves might be composed of other smaller building blocks (units). The different parts in the system are categorized as belonging to one of these units.

operating on the system, the job sheets capture information pertaining to the maintenance, diagnostics, and repairs. More specifically, they contain information about the customer complaints recorded as *calls*, the parts replaced, and high-level textual description of the work (jobs) performed by the FSE. Each call is characterized by attributes such as the *call id* (a unique identifier for the call), *customer information*, *system information* (the details of the system), *complaint* (textual description of the problem), *call open date* (the date when the problem is reported and the call is opened), *call close date* (the date when the issue is resolved and the call is closed), *number of hours* (the billable time spent in resolving the issue), and *cost* (the total cost (personnel cost, travel cost, and part(s) cost)). Each call may result in one or more *jobs*, e.g., there could be one job created for fault isolation (diagnostics), one job for replacing the part, and another job for calibration and configuration. In addition to the calls related to customer complaints, the job sheets database also contain information about jobs performed proactively such as planned maintenance and software upgrades.

Parts that can be replaced in the system are called as *Field Replaceable Units* (FRUs). In this case study, *we confine ourselves to the task of finding symptomatic patterns in the event logs that can be associated to a malfunction requiring the replacement of a FRU.* In the next two sections, we look at the data preparation process for discovering these symptomatic patterns.

9.3.2 Relating Job Sheets and System Logs

The data selection process starts with first choosing a part (FRU) of interest for which we are interested in finding the patterns, e.g., FRUs for which the variation in MTTR is large. This FRU could have been replaced in many systems as part of corrective maintenance in the past. We can identify all such systems from the job sheets database (every system across the globe is uniquely identifiable). Furthermore, each system can have multiple calls associated with this FRU replacement, i.e., it could be the case that the same part had to be replaced several times on a particular system at different periods of time. Since the system could have undergone version upgrades, it is recommended (by domain experts) that the (system, call) pairs are segregated based on their versions. Figure 9.2 depicts this scenario.

As mentioned earlier, each call is associated with a call open date and a call close date. Furthermore, the system event logs are recorded every day. For each call, we consider logs from the corresponding system a few days before the call open date and a few days after the close date for analysis. The rationale is that if there exists a symptomatic pattern, it should have manifested in the system logs prior to and during the life time of the complaint and that they disappear in the event logs after the part replacement. The number of days that one should consider before (after) the call open (close) date is largely dependent on the nature of the FRU and is to be chosen on a trial and error basis guided by some domain knowledge. For example, a malfunction in a critical part such as an X-ray tube is noticed immediately whereas a malfunction in hard disk may not be noticed immediately. Thus it may be sufficient

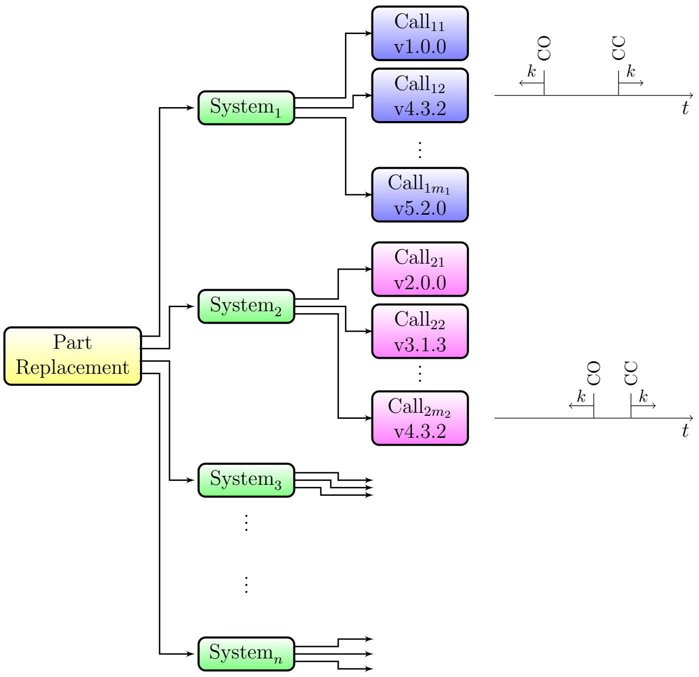


Figure 9.2: Selection of system event logs for signature discovery. For a chosen part, all systems where that part had been replaced in the past are identified. The part could have been replaced in a system several times over a period of time. Each instance of replacement is captured by a unique call. Event logs from a few days (here, k) before/after the call open (CO) and call close (CC) date for each call pertaining to a particular system version (here, v4.3.2) are chosen for analysis.

to consider just a couple of days before/after the call open/close date for the X-ray tube while for the hard disk, a larger time window is recommended.

Having discussed about the selection of system logs for analysis, the next step is to prepare the data for analysis. This requires the definition of cases (traces) and a class label for each case as illustrated in Figure 9.1. In the next section, we discuss on generating this from the selected system logs.

9.3.3 Defining Cases–Scoping and Filtering

As mentioned earlier, there are different modes during the operation of an iXR system, viz., start-up, shut-down, normal, warm-restart, and field service. Events are logged in all the system modes. Operations meant for regular use by the clinicians are logged under the normal operation mode while the operations specific to field service engineers are logged under the field service mode (this is similar to the administrator mode in an operating system). During a single day, the system could have been (re)started or shutdown multiple times. A session of system's operation constitutes the sequence of events during the normal operation mode between startup and shutdown as illustrated in Figure 9.3. Sessions form the basis for defining a case.

Before we present means of defining a case based on sessions, we briefly present the filtering of events.

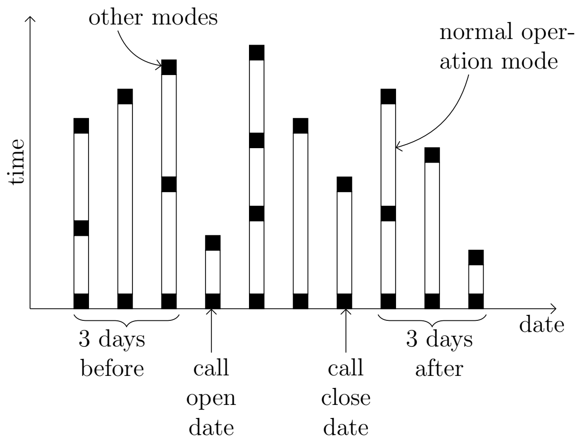


Figure 9.3: Events during normal operation mode signify the events during the regular usage of the system and constitute the focus of analysis. The system could have been restarted multiple times during a single day. Each sequence of events during the normal operation mode surrounded by other system modes defines a session. The example illustrates the consideration of log files 3 days before/after the call open/close date of a part replacement in a particular system.

The events that are recorded in the iXR system are very fine-grained. This makes the total number of events that are logged in a single day/session quite large, in the order of a few thousands. Identifying the symptomatic patterns pertaining to the malfunction of a FRU in the fine-grained event logs log is a challenging task. This can be attributed to the fact that the events that potentially bear an indication of the abnormality are just a few in number. Considering the whole log can induce a huge amount of unrelated events thereby making the task of signature discovery analogous to searching for a needle in a haystack. Domain experts suggest that a malfunction in a FRU reflects as error and/or warning events in the log pertaining to the component (unit) it belongs to and/or components with which it interacts with. Accordingly, we pre-process the log as follows:

- for a given FRU for which we are interested in identifying the symptomatic patterns in the log, we first identify (based on domain knowledge) the units (components) that are related to the FRU, e.g., if X-ray tube is chosen as the FRU, the units related to this are X-ray Control, X-ray Generator, and Geometry.
- only the error/warning events pertaining to the units related to the FRU during a session are considered

As mentioned earlier, the symptomatic patterns are expected to have manifested in the system logs prior to and during the life time of the complaint, i.e., on/before the call close date, and disappear in the event logs after the part replacement (call close date). Therefore, our problem of signature discovery is to uncover patterns consistent across the different calls (pertaining to that part replacement), which appear only in the event logs prior to the corresponding call close dates and disappear in the event logs after the call close date. It is important to note that the manifestation of patterns pertaining to a problem occurs only when that functionality or behavior is invoked on the system. In other cases, we see a normal behavior of the system. Hence in the time-period prior to the call-close date (i.e., the time period between which the customer sees some abnormality and the time at which the problem is supposed to have been resolved), it is quite possible that the system reflects a normal behavior during some of the sessions. However, the sessions that exhibit normal behavior is not known.

We propose two ways of transforming the system logs to labeled cases. For this case study, we expect the cases to be labeled as normal (N) and faulty (F). The first approach, called *individual sessions* approach, translates each session (in the system logs pertaining to a single call) into a separate case. All sessions subsequent to the call close date are labeled as normal cases. However, the sessions on/before the call close date are labeled as 'unknown' (?) at this point. The class labels for these cases are assigned later in the signature discovery framework (cf. Section 9.4). Figure 9.4 depicts this approach. The second approach, called *juxtaposed sessions* approach creates two cases per call. The sessions on/before the call close date are all appended into a single case with a distinct delimiter between them and labeled as faulty (F). Similarly, the sessions after the call close date are appended together with a distinct delimiter between them and labeled as Normal (N). The distinct delimiter is essential to ensure that patterns do not overlap across sessions during the discovery process. Figure 9.5 depicts this approach.

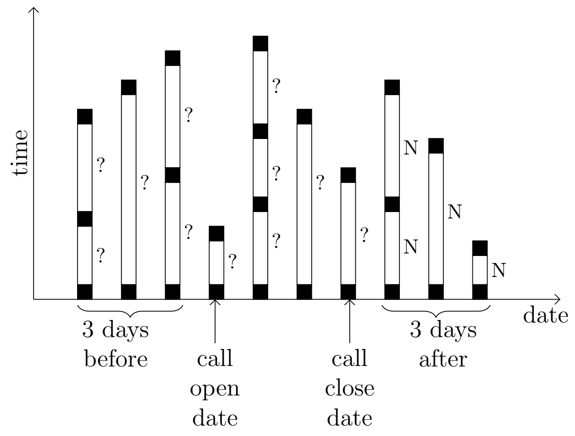


Figure 9.4: Scenario where each individual session defines a case. The cases after the call close date can be considered to be normal where as the cases on or before the call close date can either be normal or faulty depending on whether the functionality pertaining to the broken part is invoked or not. The labels for these cases are set 'unknown' (?) at this moment. The actual labels for these cases are decided later by the framework.

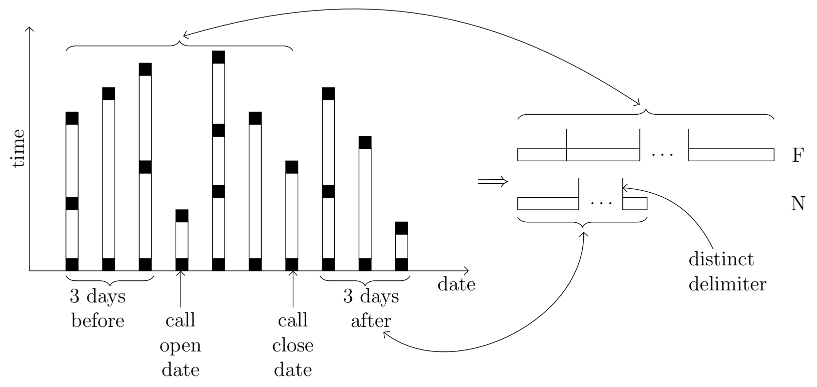


Figure 9.5: Scenario where each call defines two cases. The sessions on/before the call close date are juxtaposed and assigned the label 'faulty' (F) whereas the case defined by the juxtaposed sessions after the call close date can be considered to be 'normal' (N). Delimiters are used to distinguish the boundaries between sessions.

So far, we only discussed the data selection and preparation process. At the end of this process, we have an event log with cases that carry a class label indicating a particular class of behavior. We now proceed to uncovering the signature patterns in the next section.

9.4 Signature Discovery Framework

We propose the framework depicted in Figure 9.6 for discovering patterns that discriminate between different classes of behavior. The proposed framework is generic and works for any event log with labeled cases (signifying different classes of behavior) with a provision for some of the cases remaining unlabeled. We now explain the constituents of the framework.

Class Labeling

When event logs contain some cases that are unlabeled, an important question to address is *how can we assign labels to those unlabeled instances?*. Efficient means to automatically or semi-automatically derive labels need to be designed. We propose the use of clustering and/or classification techniques, such as the k-nearest neighbor [46] and one-class support vector machines (SVM) [200], in machine learning to assist in class labeling. For example,

- if the unlabeled instances are to be assigned one of the class labels already present in the event log, then one may consider the k-nearest neighbor approach. The basic idea is to determine the k-nearest labeled instances for each of the unlabeled instances and assign the majority class of the k instances as the class label for the unlabeled instance. The techniques proposed in Chapter 4 can be used to estimate the distance/similarity between instances.

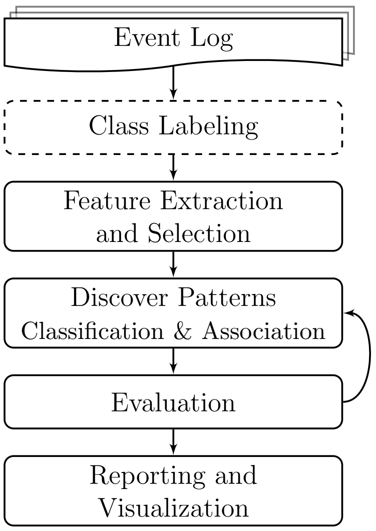

Event Log
Class Labeling
Feature Extraction and Selection
Discover Patterns
Classification & Association
Evaluation
Reporting and Visualization

Figure 9.6: Framework for signature discovery. The block depicted in dashed rectangle is an optional step that is to be considered when some of the cases in an event log are unlabeled.

- if the labeled instances in the event log belong to only one class and we are interested in labeling the unlabeled instances to utmost two-classes, e.g., normal and faulty as in the case of iXR systems, an interesting approach is the use of one-class support vector machines. Here, we assume that the instances of one-class (e.g., normal) are labeled. One-class SVMs work with the assumption that all positive (normal) instances are alike while each negative (faulty) instance can be negative in its own way, i.e., the distribution of the negative instances is unknown. Once a one-class SVM is built over the normal instances, any unlabeled instance can be evaluated to either belonging to the normal class or not and labeled accordingly.

After the execution of this step, *all instances in the event log should have a class label.* We shall now proceed to discover patterns that are specific for each class and discriminatory between the classes.

Feature Extraction and Selection

This step corresponds to extracting the features from an event log, which form the basis for signature patterns. Once features are defined, each instance in the event log is to be transformed into a vector space where the elements of the vector correspond to the value of the selected feature in the instance. We argue that a wide variety of feature types need to be considered and the choice of the feature type largely depends on the nature of the problem and its manifestation in the event log. Domain knowledge can assist us in selecting an appropriate feature. We recommend the consideration of individual events, sequence features, and alphabet features defined in Section 4.2 as features. Sequence features are important when an occurrence of a particular sequence of events in the system log defines a symptomatic pattern, e.g., when a FRU malfunctions, the units that depend on/interact with this faulty FRU *retries* and seeks for a response from the FRU. Retries often manifest as *loops*, which are captured with tandem arrays. As discussed in Section 4.2, alphabet features are derived from sequence features by relaxing the ordering of events. Sequence features that are defined over the same set of events are considered to be equivalent under an alphabet feature. In addition to the above features, one may also consider features catering to other perspectives such as data (e.g., data objects and their values in each trace).

If the number of features extracted is large, then it leads to the problem of *curse of dimensionality* [104]. Feature selection techniques deal with removing irrelevant and redundant features. One can adopt simple filtering techniques such as removing infrequent features to advanced dimensionality reduction techniques such as principal component analysis (PCA) [119] for feature selection. Once the feature extraction and selection is done, we transform the event log into a vector space as depicted in Table 9.1.

Table 9.1: The labeled cases in an event log are transformed into a vector space based on the chosen features $(f_1, f_2, \dots, f_m)$. One can choose between a nominal (binary) representation (where the value for a feature in a case corresponds to the presence/absence of the feature in that case) and a numeric representation (where the values correspond to the frequency of the feature in the case).

Instance	f_1	f_2	$\dots$	f_m	Class
1	3	1	$\dots$	0	**N**
2	0	6	$\dots$	1	**F**
3	1	0	$\dots$	4	**F**
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
n	2	2	$\dots$	0	**N**

Discover Patterns

Given a dataset as depicted in Table 9.1, the goal of this step is to discover the patterns over the features, which are strongly correlated to the class label (e.g., normal or faulty). We adopt two of the classical data mining techniques, viz., the decision trees [176, 177] and association rule mining [5, 138]. These two learning algorithms are chosen primarily for three reasons:

- they are non-parametric, i.e,. no specific data distribution (of the input dataset) is assumed
- they generate simple, understandable rules that are easy to interpret by domain experts
- they can easily handle imbalanced datasets, i.e., datasets where the instances of each class are not approximately equally represented

For the association rule mining, we adopt the special subset called the *class association rules* [138], which is an integration of classification rule mining and association rule mining. We do not present the details of these algorithms in this thesis. The interested reader is referred to [5, 104, 138, 176, 177].

The result of this step are rules of the form of disjunction (OR) of conjunctions (AND) such as

If $f_1 \geq v_{11}$ AND $f_3 = v_{31}$ AND $f_7 = v_{72}$ then **F**
OR
If $f_2 = v_{26}$ AND $f_4 = v_{47}$ then **N**
OR
If $f_5 = v_{50}$ then **F**

where the v_{ij}'s are the values for the corresponding features.

Note: If the cases in the fault diagnosis of iXR systems are defined as juxtaposed instances of sessions, then we need to ensure that the rules formed comprise of feature constraints that do not span over multiple sessions. For example, consider the case depicted in Figure 9.7 where the definition of a case is based on the juxtaposition of sessions. Let f_1 and f_3 be features manifested in one session and f_2 and f_7 be features manifested in another session. We need to ensure that we do not form rules that have a combination of say, f_3 and f_2 together. However, one may form rules that involve f_1 and f_3 or f_2 and f_7 together. This is because we are interested in finding patterns that manifest in a single session.

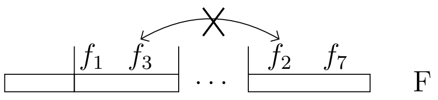


Figure 9.7: Signatures with feature combinations that overlap across multiple sessions are prohibited.

Evaluation

We adopt standard metrics in data mining such as the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), and derived metrics from these such as accuracy, sensitivity, specificity, precision, and F1-Score to evaluate the goodness of the discovered signatures. Models with sensitivity and specificity both closer to 1.0 are preferred. The reader is referred to [104] for an explanation of these metrics.

For a given dataset, one can build many classifiers. The differences mainly stem from the choice of parameter values for the learning algorithm (e.g., split criterion in decision trees, minimum support and minimum confidence constraints in association rule mining, etc.). An important characteristic of any learned model is its *generalizability*. Generalization refers to the performance of a learned model over unseen examples [104]. If the entire dataset is used for learning the signatures, then there is a danger of the uncovered signatures to be *overfitting*, i.e., the scenario where the learned model performs extremely well on the input dataset but performs poorly on unseen examples. Therefore, we adopt the cross-validation techniques during the learning phase in the above step.

Cross-validation [104, 128] is a model selection technique where the input dataset is divided into two subsets, viz., a *training set* and a *validation set*. The model is learned on the training set and evaluated on the validation set. A special case of cross-validation is the k-folds cross validation technique where the input dataset is split into k subsets, and the model is learned on the training data comprising of $k-1$ subsets and validated on the last subset. This is repeated k times with k different splits between the training and validation data. The cross-validation performance is the average of the results (of the metrics such as accuracy) on all the splits. We prefer signature patterns with a better cross-validation performance. If the performance is not satisfactory, one may change the parameter settings for the learning algorithm and re-learn the signatures.

Reporting and Visualization

The last step in the framework deals with the reporting and visualization aspects. Automated reports eliciting the signature patterns along with their performance metrics are generated. Apart from reports, one may depict the results in pictorial forms such as pie-charts and scatter plots. For example, Figure 9.8 depicts the projection of a two-class multi-dimensional data onto the top two principal components obtained using the principal component analysis (PCA). Such a visualization helps in assessing the goodness of a feature set. In the figure, we can see that the two classes (normal and faulty) are clearly separable thereby indicating that the chosen feature set representation for the cases is good enough to find discriminatory patterns.

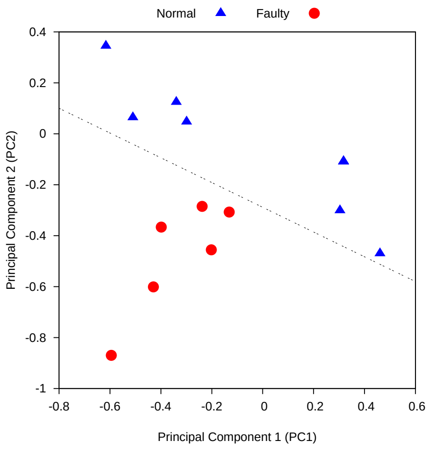


Figure 9.8: Visualization of dataset using principal components.

9.5 Experiments and Discussion

In this section, we discuss the application of the proposed framework in uncovering signatures for two of the FRUs, which we anonymize as FRU I and FRU II, in the iXR systems. There are different types of iXR systems, e.g., monoplane, biplane, etc. They are uniquely identified by a system code, e.g., 722001, 722002, etc. We analyzed the logs for one of the system types. First, we discuss the results for FRU I. For learning the signature patterns, we considered this FRU replacements that happened in the years 2008 and 2009. As discussed earlier, since different versions of the system can have different signatures, we split the systems according to their versions and discover the signatures for each version separately[3]. From the jobsheets, we identified the systems and the dates when this FRU was replaced and selected system log files between three days before the call open date and three days after the call close date for these systems. We considered the error/warning events from three units, which we anonymize (for confidentiality reasons) as Unit A, Unit B, and Unit C. These three units are considered to be the most relevant for this FRU by the domain experts. We used the juxtaposed sessions approach (cf. Section 9.3.3) for defining the cases and class labels. Using this procedure, we created cases with two labels, i.e., normal or faulty, for each system type and version, e.g., there are 32 instances for the system type 722006 and system version 4.3.5.

We discovered the signature patterns from these instances using the framework described in Section 9.4. We used a combination of tandem repeat alphabet and maximal repeat alphabet features (cf. Section 4.2) in conjunction with the class association rules for learning the signature patterns. A couple of example signatures for the faulty class are provided in Table 9.2. The two signatures differ in the last two events. The interpretation for this is that this FRU has multiple failure modes and the manifestation of failure modes differ in the system event logs. Each of the signatures in Table 9.2 captures one of these failure modes.

Table 9.2: Example signature patterns for FRU I.

If	`xxxxxxxx1` Warning from `Unit A` is Present AND `xxxxxxxx2` Error from `Unit A` is Present AND `xxxxxxxx4` Warning from `Unit B` is Present AND `xxxxxxxx4` Warning from `Unit A` is Present AND `xxxxxxxx5` Error from `Unit A` is Present	Then	Faulty
If	`xxxxxxxx1` Warning from `Unit A` is Present AND `xxxxxxxx2` Error from `Unit A` is Present AND `xxxxxxxx4` Warning from `Unit B` is Present AND `xxxxxxxx6` Warning from `Unit A` is Present AND `xxxxxxxx7` Error from `Unit A` is Present	Then	Faulty

We have evaluated the goodness of the signature patterns on an independent test

[3]It could be the case that for two different versions, the signatures are the same (this implies that there is no change between these versions with respect to this FRU)

set of system logs between Jan 2010 and Jun 2011 (Note that the signatures were discovered using logs from 2008 and 2009). Signatures of a particular system type and version are evaluated against systems of the same type and version. Table 9.3 depicts the performance of the signatures for two different versions of systems. The interpretation of the true positives, false positives, true negatives, and false negatives are as follows:

- *TP:* the signature is present in one or more log instances[4] of a system AND there is a FRU replacement in the system subsequently AND the signature disappears after the replacement
- *FP:* the signature is present in one or more log instances of a system BUT there is no FRU replacement in the system (OR) the signature is present in one or more log instances of a system AND there is a FRU replacement in the system subsequently BUT the signature does not disappear after the replacement
- *TN:* the signature is not present in a log instance of a system AND there is no FRU replacement in this system
- *FN:* the signature is not present in a log instance of a system BUT there is a FRU replacement in the system.

False Negatives indicate that the discovered signatures are not complete and that there might be other symptomatic patterns which we are not able to uncover. This results when the training data does not represent all manifestations of failure modes of the FRU. False Negatives are not devastating whereas false positives are. False Positives have serious repercussions and need to be minimized. False positives can lead to false replacements. False negatives affect the sensitivity and F1-Score metrics while false positives affect the specificity, precision, and F1-Score metrics.

From Table 9.3, we can see that the uncovered signatures perform quite well with a very high accuracy (above 98%). Note that our evaluation involved a large set of independent systems (743 in number) with logs considered in a different time period from that of the training data. The uncovered signatures for version 3.1.7 are able to detect all but one of the required replacements and there are no false positives. The discovered signatures for version 4.3.5 are able to indicate a problem in this FRU for 24 of the 32 replacements and could not capture the rest 8 replacements. The part could have exhibited a failure mode different from that of the captured signatures in these 8 replacements. Furthermore, we see just 2 false positives in this case.

As another example, we considered a different FRU, anonymized as FRU II. Just like in the previous scenario, we considered part replacements in 2008 and 2009. Event logs from systems with this part replacement were used for learning the signature patterns. Signatures discovered using the class association rule mining algorithm on the individual events (as the feature set) performed better when compared to other features and learning algorithms on the training data. We evaluated the uncovered signatures on an independent set of system logs between Jan 2010 and Jun 2011. Table 9.3 depicts the performance of the discovered signatures for two different

[4]a log instance is one session of the system log in the normal operation mode

Table 9.3: Performance of the discovered signatures for the two FRU replacements on an independent test set of systems. Various metrics such as the true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), and derived metrics such as accuracy, sensitivity, specificity, precision, and F1-score are measured.

Part	Version	No. Systems	TP	FP	TN	FN	Accuracy $\frac{TP+TN}{\text{No.Systems}}\%$	Sensitivity (r) $\frac{TP}{TP+FN}\%$	Specificity $\frac{TN}{TN+FP}\%$	Precision (p) $\frac{TP}{TP+FP}\%$	F1-Score $\frac{2*p*r}{p+r}\%$
FRU I	3.1.7	120	3	0	116	1	99.16	75.00	100.00	100.00	85.71
	4.3.5	623	24	2	589	8	98.39	75.00	99.66	92.30	82.75
FRU II	3.1.7	120	7	0	112	1	99.16	87.50	100.00	100.00	93.33
	4.3.5	623	47	2	564	10	98.07	82.45	99.64	95.92	88.67

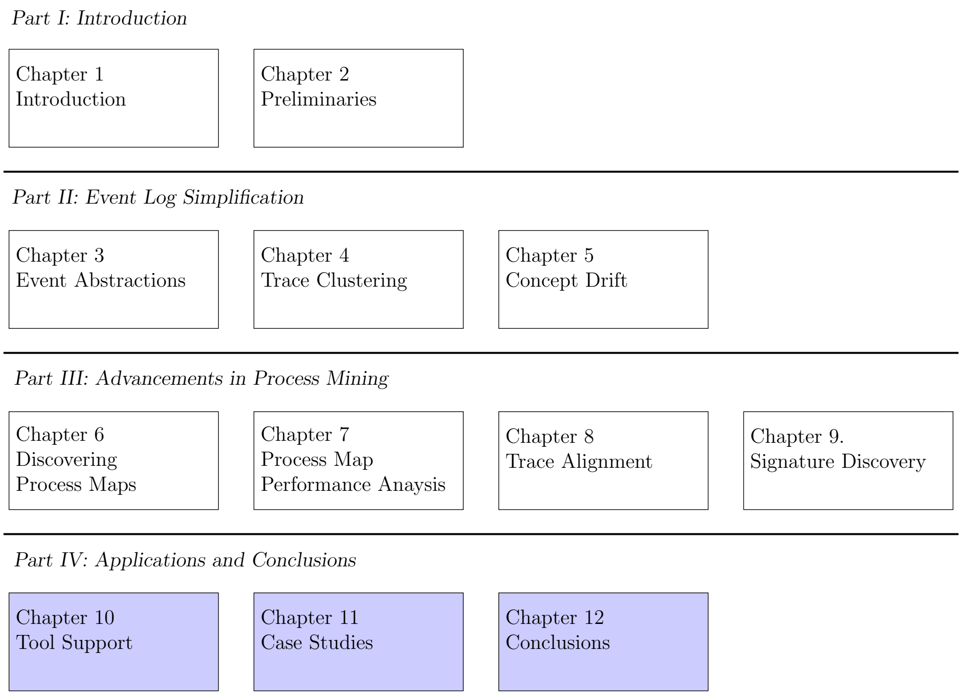
versions of systems. We can see that, even in this case, the uncovered signatures perform pretty decently with high accuracy (above 98%). The uncovered signatures for version 3.1.7 are able to detect all but one of the required replacements and there are no false positives. As mentioned before, false negatives potentially indicate different failure modes, the signatures of which are not captured in our discovery phase, primarily due to lack of representative instances for this failure mode in the training phase. Systems of version 4.3.5 had a larger number of tube replacements and the discovered signatures are able to detect a problem in the tube in 82% of the cases (sensitivity metric). The discovered signatures resulted in only 2 false positives. The discovered signatures could not cover some of the failure modes and this is reflected in the 10 false negatives. As mentioned earlier, false negatives can be lived with. The signatures for these failure modes also can be discovered when event logs representing these failure modes are provided in the training dataset. To summarize, the proposed framework for signature discovery shows significant promise in fault diagnosis of X-ray machines.

The discovered signatures can be added to a knowledge base and the logs of systems scanned for the presence of signature patterns. This can assist the FSEs during their diagnostic efforts. The FSEs can be provided with log analyzer tools that checks for the presence of signatures. The manifestations of signature patterns suggest potential problematic parts (FRUs) corresponding to the signature. Since diagnostics are considered to be the most time consuming and the most difficult task, such an automated assistance is expected to reduce the MTTR significantly.

9.6 Conclusions

In this chapter, we explored the feasibility of automatically identifying signature patterns that can discriminate between different classes of behavior. We demonstrated that the proposed framework works well, i.e., it is able to uncover signatures with a high accuracy, by applying it over two real-life case studies on X-ray machines. The resulting signatures remain highly accurate even on unseen instances. This indicates that the suggested framework has the potential to become a powerful tool for the diagnosis of failures in iXR systems. The proposed framework is generic and can be applied in other similar environments. We have used two of the classical algorithms, viz., decision trees and association rule mining, for learning the signatures. However, the framework can easily be extended with other discovery algorithms.

Part IV
Applications and Conclusions


Part I: Introduction
Chapter 1 Introduction
Chapter 2 Preliminaries
Part II: Event Log Simplification
Chapter 3 Event Abstractions
Chapter 4 Trace Clustering
Chapter 5 Concept Drift
Part III: Advancements in Process Mining
Chapter 6 Discovering Process Maps
Chapter 7 Process Map Performance Anaysis
Chapter 8 Trace Alignment
Chapter 9. Signature Discovery
Part IV: Applications and Conclusions
Chapter 10 Tool Support
Chapter 11 Case Studies
Chapter 12 Conclusions

Chapter 10 presents the tool support implemented as plug-ins in the ProM framework, which realizes the techniques presented in the previous parts of this thesis. Chapter 11 presents a few case studies illustrating the application and relevance of the techniques proposed in this thesis on real-life event logs. Chapter 12 concludes this thesis by summarizing the contributions made in this thesis and listing some of the challenges and directions for future work that need to be addressed to make process mining even more amenable for large scale event logs.

Chapter 10
Tool Support

The concepts presented in this thesis have been realized as plug-ins in the ProM[1] framework. ProM is a “plug-able” environment for process mining envisioned to provide a common basis for all kinds of process mining techniques ranging from importing, exporting, and filtering event logs (process models) to analysis and visualization of results. The initial versions of ProM (versions 1.1 to 5.2) used MXML [244] as the input format for event logs. Over years, ProM has emerged to be the *de facto standard* for process mining. The architecture of ProM has been redesigned completely when going from version 5.2 to version 6.0. The primary factors that led to this change are two fold: (a) to enable the distributed execution of plug-ins, and (b) to separate the user interface/visualization from the analysis techniques. ProM 6.0 is based on XES [92] rather than MXML. As discussed in Chapter 2, XES is the new process mining standard for event logs adopted by the IEEE Task Force on Process Mining. ProM 6.0 supports MXML as well.

The plug-ins supporting the concepts presented in this thesis are developed in the ProM 6.0 framework (the current version is 6.1; the developed plug-ins hold for all later versions as well). This chapter gives a brief overview of the different plug-ins and is organized as follows. Section 10.1 presents the Pattern Abstractions plug-in, which enables the definition of abstractions (based on common execution patterns) and the transformation of an event log to a desirable level of granularity. Section 10.2 presents the Guide Tree Miner plug-in, which assists in dealing with the heterogeneity of event logs by partitioning an event log into sets of homogenous cases. Section 10.3 presents the Concept Drift plug-in, which assists in identifying the instants of time when a process has changed (if any) and the fragments where a process could have undergone a change. Section 10.4 presents the Fuzzy Map Miner plug-in that enables the discovery of hierarchical process models. Section 10.5 presents the Fuzzy Map Performance Analysis plug-in, which assists in annotating a process map with performance metrics by replaying an event log. Section 10.6 presents the Trace Alignment With Guide Tree plug-in, which aligns the traces based on context. Section 10.7 presents the Signature Discovery plug-in, which assists in extracting signature patterns/rules that have a strong correlation/association to a particular behavior.

[1]See `www.processmining.org` for more information.

10.1 Pattern Abstractions

The Pattern Abstractions plug-in has been implemented as a *log visualizer* in ProM and caters to *the discovery of common execution patterns, the definition of abstractions over them (as defined in Chapter 3), and the pre-processing of the event log with these abstractions (as defined in Chapter 6).* The basic building blocks of the Pattern Abstractions plug-in are shown in Figure 10.1. Figures 10.2 and 10.3 illustrate these building blocks.

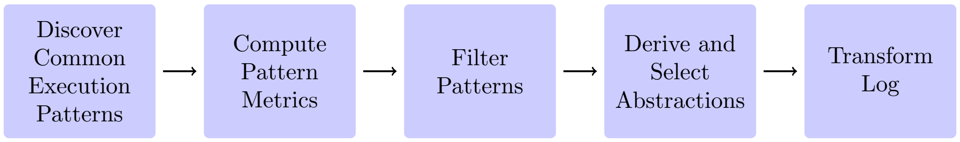
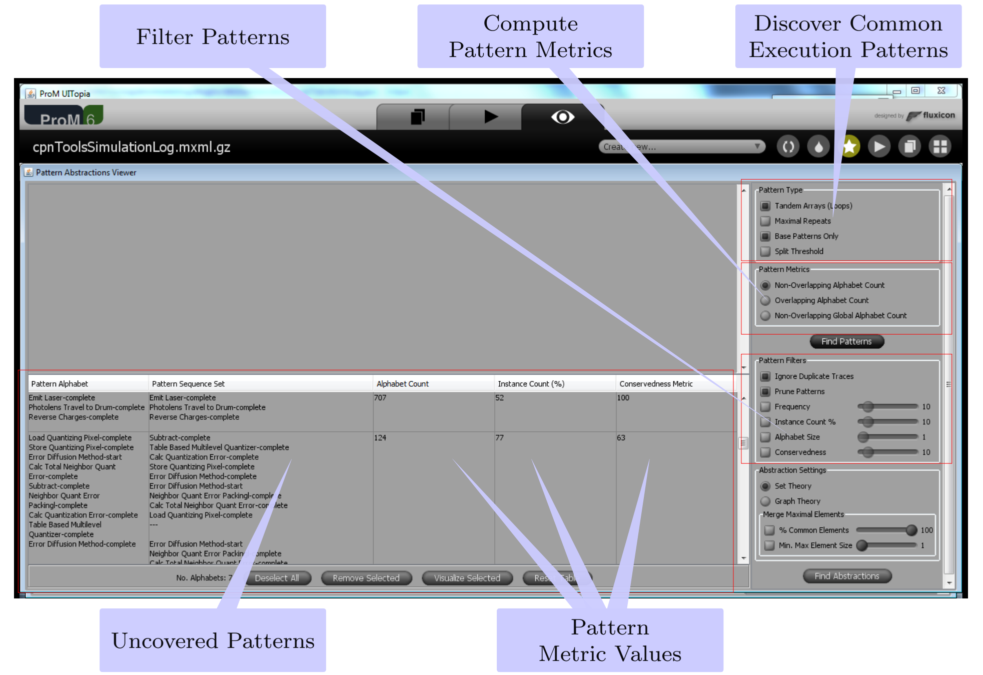

Discover Common Execution Patterns
Compute Pattern Metrics
Filter Patterns
Derive and Select Abstractions
Transform Log

Figure 10.1: Building blocks of the Pattern Abstractions plug-in.

- *Discover Common Execution Patterns:* The Pattern Abstractions plug-in supports the discovery of tandem arrays (loop patterns) and maximal repeats (common subsequence of activities within a process instance or across process instances) (cf. Chapter 3). These can be uncovered in *linear* time and space with respect to the length of the traces.
- *Compute Pattern Metrics:* Various metrics (e.g., overlapping and non-overlapping frequency counts, instance percentage, etc. as defined in Chapter 3) to assess the significance of the uncovered patterns are supported.
- *Filter Patterns:* It could be the case that too many patterns are uncovered from the event log. To manage this, features to filter patterns that are less significant are supported.
- *Derive and Select Abstractions:* Abstractions are defined over the filtered patterns. Patterns that are closely related are grouped together to form abstractions. The approach for forming abstractions is presented in Chapter 3. Furthermore, various features to edit/select abstractions such as merging two or more abstractions and deleting activities related to a particular abstraction are supported. Figure 10.3 depicts a few abstractions defined over loop patterns for the copier event log, e.g., Half Toning, a procedure for enhancing the image quality, is uncovered as an abstraction.
- *Transform Log:* The event log is pre-processed by replacing activity subsequences corresponding to abstractions. A replaced activity subsequence is captured as a process instance in the sub-log for the corresponding abstract activity. This realizes the approach presented in Chapter 6. If n abstractions are selected, the Pattern Abstractions plug-in generates a transformed log, and n sub-logs (one for each of the n chosen abstractions).

The Pattern Abstractions plug-in supports additional features such as visualizing patterns and exporting the traces that contain the patterns.



cpnToolsSimulationLog.mxml.gz

Pattern Abstractions Viewer

Pattern Type
- Tandem Arrays (Loops)
- Maximal Repeats
- Base Patterns Only
- Split Threshold

Pattern Metrics
- Non-Overlapping Alphabet Count
- Overlapping Alphabet Count
- Non-Overlapping Global Alphabet Count

Pattern Filters
- Ignore Duplicate Traces
- Prune Patterns
- Frequency 10
- Instance Count % 10
- Alphabet Size 1
- Conservedness 10

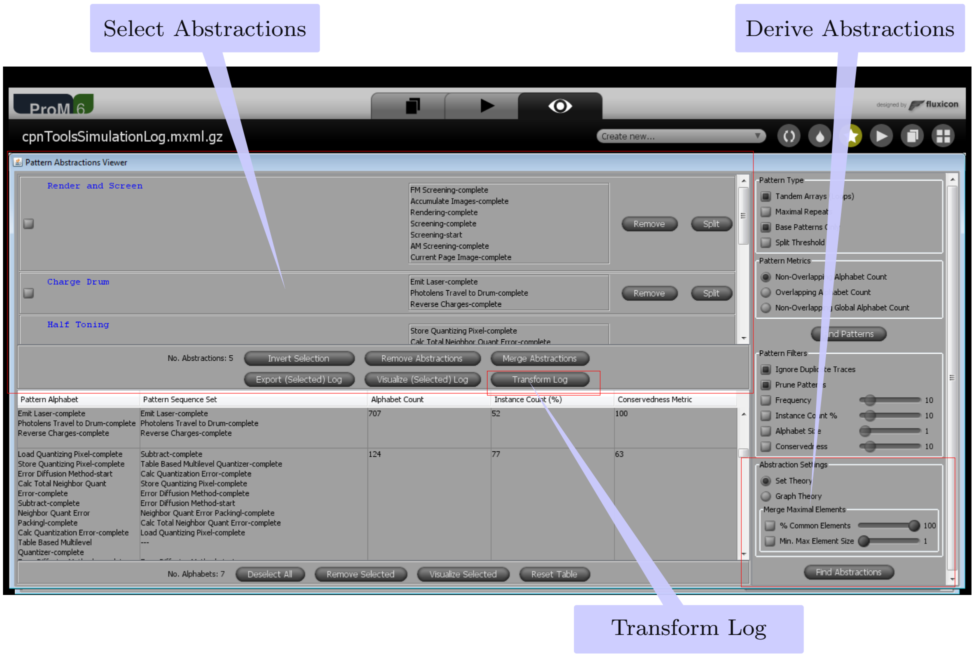
Abstraction Settings
- Set Theory
- Graph Theory

Merge Maximal Elements
- % Common Elements 100
- Min. Max Element Size 1

Pattern Alphabet	Pattern Sequence Set	Alphabet Count	Instance Count (%)	Conservedness Metric
Emit Laser-complete Photolens Travel to Drum-complete Reverse Charges-complete	Emit Laser-complete Photolens Travel to Drum-complete Reverse Charges-complete	707	52	100
Load Quantizing Pixel-complete Store Quantizing Pixel-complete Error Diffusion Method-start Calc Total Neighbor Quant Error-complete Subtract-complete Neighbor Quant Error Packingl-complete Calc Quantization Error-complete Table Based Multilevel Quantizer-complete Error Diffusion Method-complete	Subtract-complete Table Based Multilevel Quantizer-complete Calc Quantization Error-complete Store Quantizing Pixel-complete Error Diffusion Method-complete Error Diffusion Method-start Neighbor Quant Error Packingl-complete Calc Total Neighbor Quant Error-complete Load Quantizing Pixel-complete --- Error Diffusion Method-start Neighbor Quant Error Packingl-complete Calc Total Neighbor Quant Error-complete	124	77	63

No. Alphabets: 7

Figure 10.2: The discovery of common execution patterns, computation of pattern metrics, filtering and inspection of patterns in the Pattern Abstractions plug-in.



cpnToolsSimulationLog.mxml.gz

Pattern Abstractions Viewer

Render and Screen
- FM Screening-complete
- Accumulate Images-complete
- Rendering-complete
- Screening-complete
- Screening-start
- AM Screening-complete
- Current Page Image-complete

Charge Drum
- Emit Laser-complete
- Photolens Travel to Drum-complete
- Reverse Charges-complete

Half Toning
- Store Quantizing Pixel-complete
- Calc Total Neighbor Quant Error-complete

No. Abstractions: 5

Pattern Alphabet	Pattern Sequence Set	Alphabet Count	Instance Count (%)	Conservedness Metric
Emit Laser-complete Photolens Travel to Drum-complete Reverse Charges-complete	Emit Laser-complete Photolens Travel to Drum-complete Reverse Charges-complete	707	52	100
Load Quantizing Pixel-complete Store Quantizing Pixel-complete Error Diffusion Method-start Calc Total Neighbor Quant Error-complete Subtract-complete Neighbor Quant Error Packingl-complete Calc Quantization Error-complete Table Based Multilevel Quantizer-complete	Subtract-complete Table Based Multilevel Quantizer-complete Calc Quantization Error-complete Store Quantizing Pixel-complete Error Diffusion Method-complete Error Diffusion Method-start Neighbor Quant Error Packingl-complete Calc Total Neighbor Quant Error-complete Load Quantizing Pixel-complete ---	124	77	63

No. Alphabets: 7

Figure 10.3: The generation and selection of abstractions in the Pattern Abstractions plug-in.

10.2 Guide Tree Miner

The Guide Tree Miner plug-in implements the concepts presented in Chapter 4 on trace clustering. The basic building blocks of the plug-in are shown in Figure 10.4.

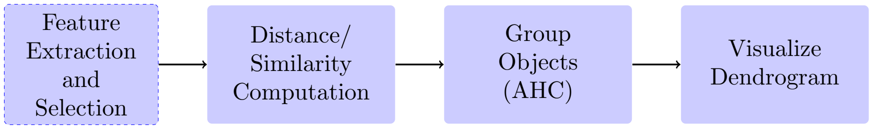
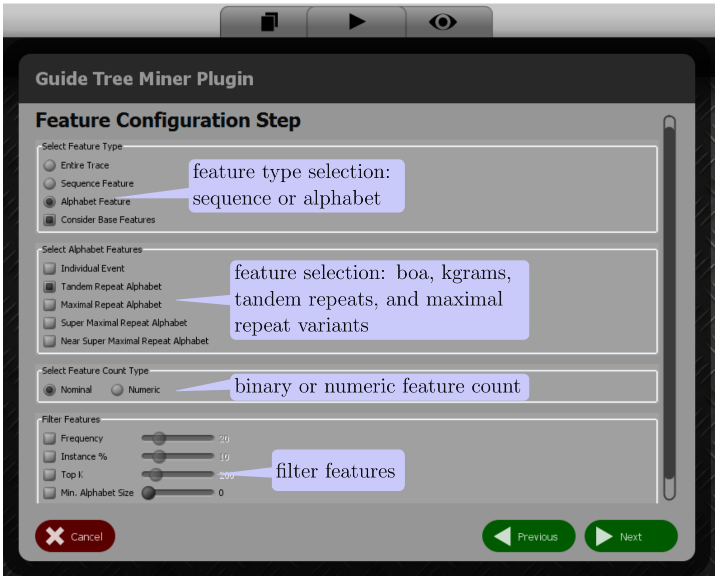

Feature Extraction and Selection
Distance/ Similarity Computation
Group Objects (AHC)
Visualize Dendrogram

Figure 10.4: Building blocks of the Guide Tree Miner plug-in. Feature extraction and selection step is optional and is applicable only for vector-based approaches to trace clustering.

- *Feature Extraction and Feature Selection:* The plug-in supports both the sequence and alphabet features presented in Chapter 4 for the vector-based approaches. The traces in an event log can be transformed into a vector space using either the binary or numerical representation. Furthermore, the plug-in supports options for filtering features based on their frequency, percentage of instances in which they manifest, and alphabet size. Figure 10.5 depicts the feature configuration step for the vector-based approaches.
- *Distance or Similarity Computation:* The plug-in supports different methods for computing the (dis)-similarity between traces. For the vector-based approaches, standard distance metrics such as the Euclidean distance are supported. We also support a similarity measure, called F-score [62], for the vector-based approaches. For the syntactic approaches, the distance between traces can be computed using either the Levenshtein distance or the generic edit distance. The plug-in supports the derivation of the scores for substitution and indels automatically from an event log based on the procedure explained in Section 4.4. Alternatively, one can provide these scores based on domain knowledge. The plug-in loads those scores specified and uses them for computing the generic edit distance. Figure 10.6 depicts the distance configuration step for the syntactic approaches.
- *Agglomerative Hierarchical Clustering:* Once the inter-trace (dis)-similarities are computed, the next step is to group objects using the Agglomerative Hierarchical Clustering (AHC) algorithm. As discussed in Section 4.5, different join criteria can be used when grouping objects. The plug-in supports five join criteria, viz., single linkage, complete linkage, average linkage, centroid linkage, and minimum variance. Though AHC has an embedded flexibility on the level of abstraction (the number of clusters), the user can optionally specify apriori the number of clusters to partition the log into. If specified, the plug-in generates as output, k event logs (for a chosen number of clusters $k \in \mathbb{N}$), each containing the partitioned traces pertaining to that cluster. Figure 10.7 depicts the configuration step for AHC.
- *Visualization:* As discussed in Chapter 4, the hierarchy obtained using AHC can be visualized as a dendrogram. The plug-in supports the graphical depiction and exploration of the dendrogram. If the number of clusters k is specified

in the previous step, the plug-in highlights the nodes in the dendrogram corresponding to the root of the clusters. All leafs (traces) in the sub-tree at a root node pertaining to a cluster belong to that cluster. Figure 10.8 depicts the visualization of the dendrogram.


Guide Tree Miner Plugin

Feature Configuration Step

Select Feature Type: Entire Trace, Sequence Feature, [✓ Alphabet Feature]; [✓ Consider Base Features]

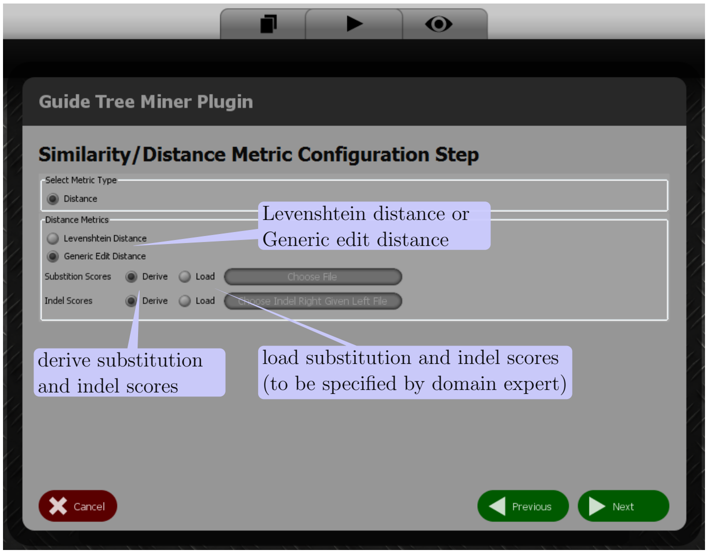
Select Alphabet Features: Individual Event, [✓ Tandem Repeat Alphabet], Maximal Repeat Alphabet, Super Maximal Repeat Alphabet, Near Super Maximal Repeat Alphabet

Select Feature Count Type: [✓ Nominal] Numeric

Filter Features: Frequency 20, Instance % 10, Top K 200, Min. Alphabet Size 0

feature type selection: sequence or alphabet
feature selection: boa, kgrams, tandem repeats, and maximal repeat variants
binary or numeric feature count
filter features
Cancel
Previous
Next

Figure 10.5: Feature configuration step for the vector-based approaches to trace clustering.


Guide Tree Miner Plugin

Similarity/Distance Metric Configuration Step

Select Metric Type: [✓ Distance]

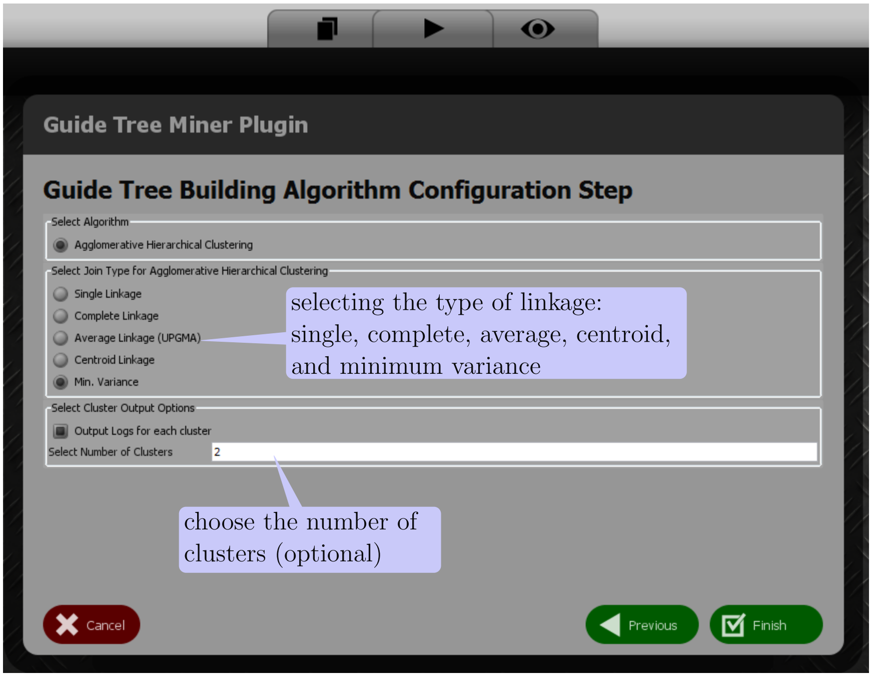
Distance Metrics: Levenshtein Distance, [✓ Generic Edit Distance]

Substitution Scores: [✓ Derive] Load — Choose File

Indel Scores: [✓ Derive] Load — Choose Indel Right Given Left File

Levenshtein distance or Generic edit distance
derive substitution and indel scores
load substitution and indel scores (to be specified by domain expert)
Cancel
Previous
Next

Figure 10.6: Configuring the distance metric in the Guide Tree Miner plug-in for the syntactic approaches. Both the Levenshtein edit distance and the generic edit distance are supported.


selecting the type of linkage: single, complete, average, centroid, and minimum variance
choose the number of clusters (optional)
Cancel
Previous
Finish

Guide Tree Miner Plugin

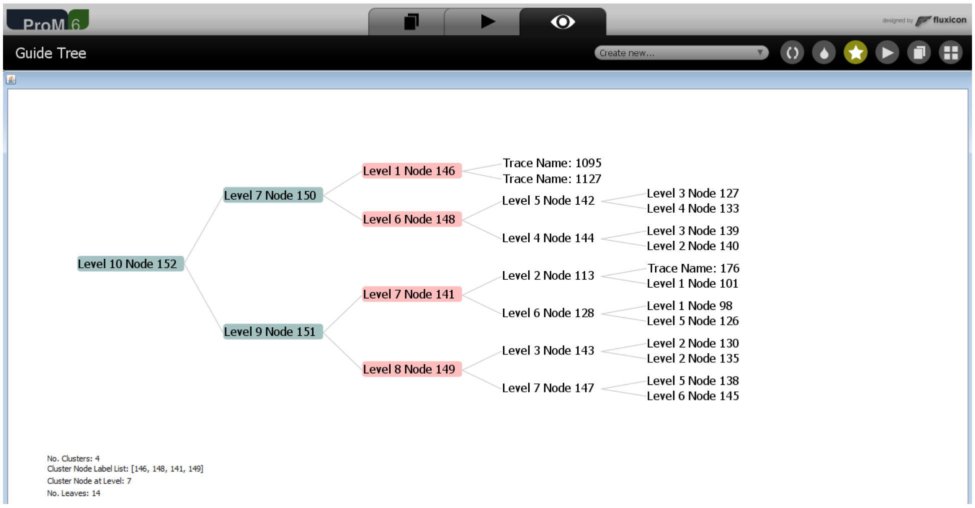
Guide Tree Building Algorithm Configuration Step

Select Algorithm
[✓ Agglomerative Hierarchical Clustering]

Select Join Type for Agglomerative Hierarchical Clustering
Single Linkage
Complete Linkage
Average Linkage (UPGMA)
Centroid Linkage
[✓ Min. Variance]

Select Cluster Output Options
Output Logs for each cluster
Select Number of Clusters: 2

Figure 10.7: Configuring the join criteria for the agglomerative hierarchical clustering algorithm.



```
No. Clusters: 4
Cluster Node Label List: [146, 148, 141, 149]
Cluster Node at Level: 7
No. Leaves: 14
```

Figure 10.8: Visualization of the mined Guide Tree. The plug-in has been configured to split the log into four clusters (the number of clusters was chosen to be four based on apriori knowledge on the process). The pink nodes represent the root of the sub-tree corresponding to the four clusters. The tree can be expanded by clicking a non-leaf node. Blue nodes indicate nodes that are expanded.

10.3 Concept Drift

The Concept Drift plug-in realizes the ideas presented in Chapter 5. Given an event log, this plug-in serves two purposes: (i) to detect whether a change has taken place and if so, at what points, and (ii) to localize the regions (process fragments) where a change could have potentially taken place. There are two configuration steps for this plug-in

- *Feature Configuration:* As discussed in Chapter 5, the detection of drifts relies on the premise that characteristic differences exist in the traces before and after the change points. The feature configuration step lets the user choose how to characterize the traces. Two scopes of features are supported: (a) global features (defined at the log or sub-log level) and (b) local features (defined at the trace level). The plug-in supports two global features (Relation Type Count and Relation Entropy) and two local features (Window Count and J-measure) (cf. Section 5.4.1). Figure 10.9 depicts the configuration step for the local features. The global features are defined over each activity in the event log and the local features are defined over every activity pair. The plug-in supports the selection of activities (or activity pairs) of interest by the user.
- *Hypothesis Test Configuration:* This step enables the selection of a hypothesis test and the population size. The plug-in supports four hypothesis tests, viz., KS-test, MW-test, T-test, and Gamma test (cf. Section 5.4.2 and refer to [202]). Figure 10.10 depicts the configuration step for hypothesis tests. The plug-in can be easily extended to support additional hypothesis tests. The plug-in uses the java statistical classes library [14] for the statistical tests.

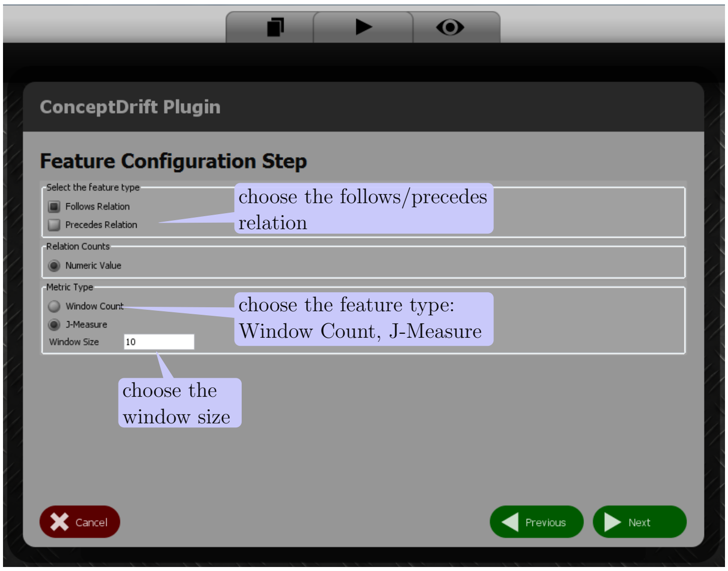
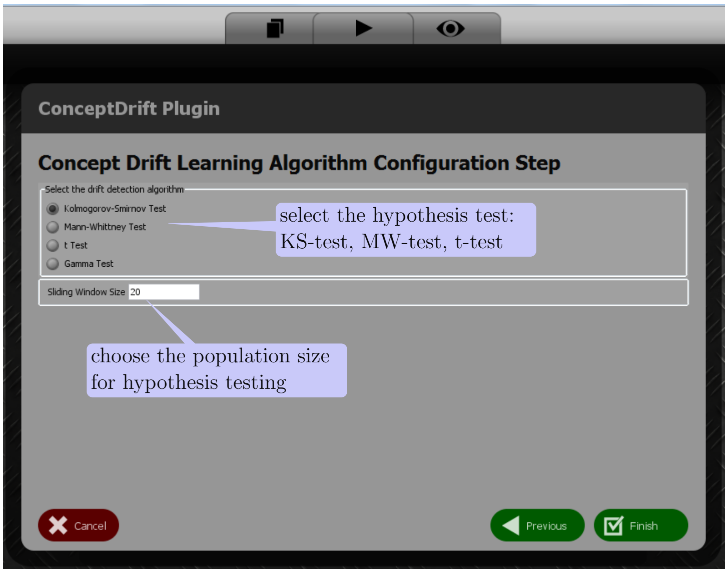


Figure 10.9: Configuration step for the local features to characterize traces.



ConceptDrift Plugin

Concept Drift Learning Algorithm Configuration Step

Select the drift detection algorithm

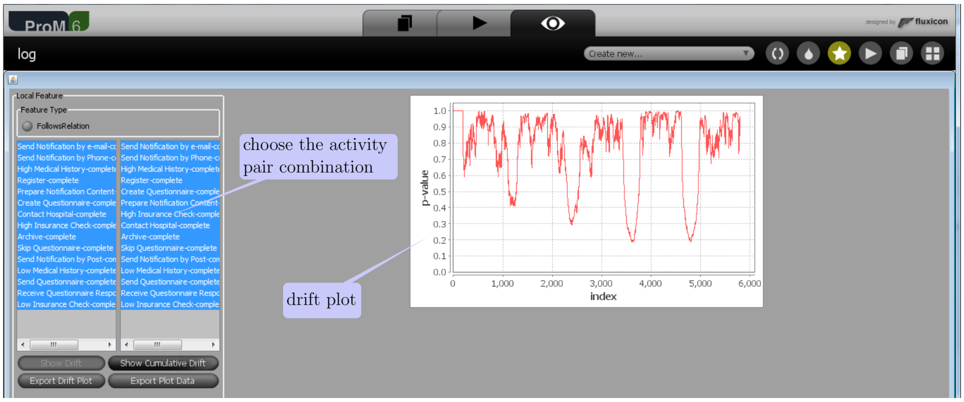
- [✓ Kolmogorov-Smirnov Test]
- Mann-Whittney Test
- t Test
- Gamma Test

Sliding Window Size 20

select the hypothesis test: KS-test, MW-test, t-test
choose the population size for hypothesis testing
Cancel
Previous
Finish

Figure 10.10: Configuration step for hypothesis testing, which comprises of choosing a statistical test and the population size.

The plug-in supports the visualization of the significance probability for the hypothesis tests as a drift plot. In order to detect the points of change, the cumulative plot depicting the average significance probability (over the selected activities/activity pairs) needs to be considered. Fragments of change (change localization) can be detected by considering the average significance probability for subsets of activities or activity pairs. The plug-in provides support for an interactive exploration of these variations. Figure 10.11 depicts an example visualization of the drift plot.



Local Feature

Feature Type

- FollowsRelation

Send Notification by e-mail-co	Send Notification by e-mail-co
Send Notification by Phone-co	Send Notification by Phone-co
High Medical History-complete	High Medical History-complete
Register-complete	Register-complete
Prepare Notification Content-	Create Questionnaire-comple
Create Questionnaire-comple	Prepare Notification Content-
Contact Hospital-complete	High Insurance Check-comple
High Insurance Check-comple	Contact Hospital-complete
Archive-complete	Archive-complete
Skip Questionnaire-complete	Skip Questionnaire-complete
Send Notification by Post-con	Send Notification by Post-con
Low Medical History-complete	Low Medical History-complete
Send Questionnaire-complete	Send Questionnaire-complete
Receive Questionnaire Respo	Receive Questionnaire Respo
Low Insurance Check-comple	Low Insurance Check-comple

Figure 10.11: Concept drift visualization. The user can get a cumulative drift plot over all activity pairs (for change detection) or over selected subsets of activity pairs (for change localization).

10.4 Fuzzy Map Miner

The two-phase approach to process discovery described in Chapter 6 enables the discovery of hierarchical process models. This can be realized using a chain of plug-ins implemented in ProM. The chain of plug-ins and their order of application is illustrated in Figure 10.12.

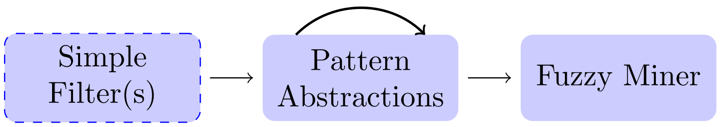


Figure 10.12: The chaining of plug-ins that enables the discovery of hierarchical process models.

The event log may first be cleansed using some simple filters (e.g., adding artificial start/end events, filtering events of a particular transaction type such as considering only 'complete' events, etc.). The Pattern Abstractions plug-in as discussed in Section 10.1 is then applied on this filtered log one or several times. The transformed log (pre-processed log with abstractions) obtained in iteration i is used as the input for the Pattern Abstractions plug-in in iteration $i + 1$. It is this repetitive application of the Pattern Abstractions plug-in that enables the discovery of multiple levels of hierarchy (new abstractions can be defined over existing abstractions as described in Chapter 6). During the pre-processing phase, for each defined abstraction, the Pattern Abstractions plug-in generates a sub-log that captures the manifestation of execution patterns defined by that abstraction as its process instances. The Fuzzy Miner plug-in [94] is then applied on the transformed log obtained after the last iteration.

The Fuzzy Miner plug-in has been implemented by several people. Günther [94] initially developed it for earlier versions of ProM (version 5.2 or older). It has been ported to version 6.0 by Li [137], Xia [270], and Eric [23, 24]. We have extended it to utilize the availability of sub-logs for the defined abstractions. Process models are discovered for each of the sub-logs and are displayed upon zooming in on its corresponding abstract activity. Abstract activities are differentiated from other activities by means of a distinct color (a darker shade of blue). We call this the Fuzzy Map Miner. Fuzzy Map Miner gives the user more control over the discovery process than the classical Fuzzy miner [94].

Figure 10.13 shows the architecture of the Fuzzy Map Miner. First, the user mines an event log for a Fuzzy Model. The user can view different perspectives of the mined model using a number of viewers, among which is the Fuzzy Model viewer. Second, the user creates a Fuzzy Instance from this Fuzzy Model by either exporting it from the Fuzzy Model viewer, or by automatically selecting the best-conforming Fuzzy Instance. Combined, this Fuzzy Instance and the original event log can be turned into a Fuzzy Animation, which can also be viewed by the user. The parameter configuration steps for the Fuzzy Map Miner remain the same as that of the classical Fuzzy miner.

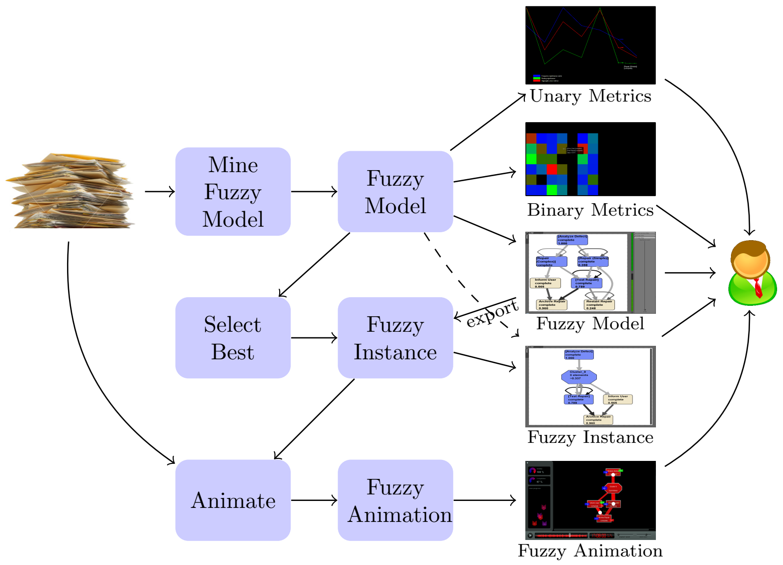


Figure 10.13: Fuzzy Map Miner Architecture.

10.5 Fuzzy Map Performance Analysis

The Fuzzy Map Performance Analysis plug-in realizes the techniques proposed in Chapter 7 for replaying an event log. During replay, various performance measures are computed. The computed measures are annotated on the Fuzzy model so that bottlenecks can be identified. This plug-in takes as input an event log and a hierarchical Fuzzy model exported from the Fuzzy Map Miner. Figure 10.14 shows the result of the plug-in. The Fuzzy model is annotated with measures such as the number of executions and average execution (throughput) time. Figure 10.15(a) depicts the configuration panel for choosing the strategy for node/edge coloring. The plug-in supports three strategies as discussed in Section 7.4. The annotated Fuzzy model can be interactively explored. Upon selecting a node/edge, the various performance metrics pertaining to that node/edge is shown in a separate panel as depicted in Figure 10.15(b).

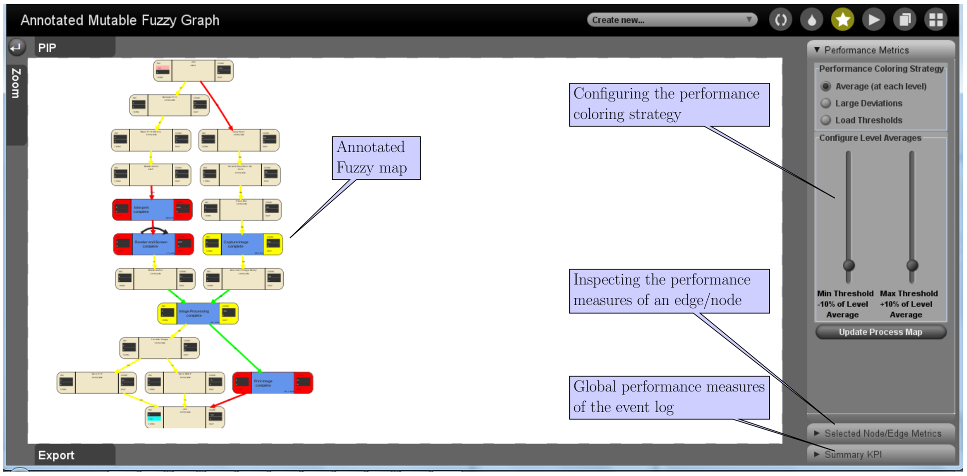
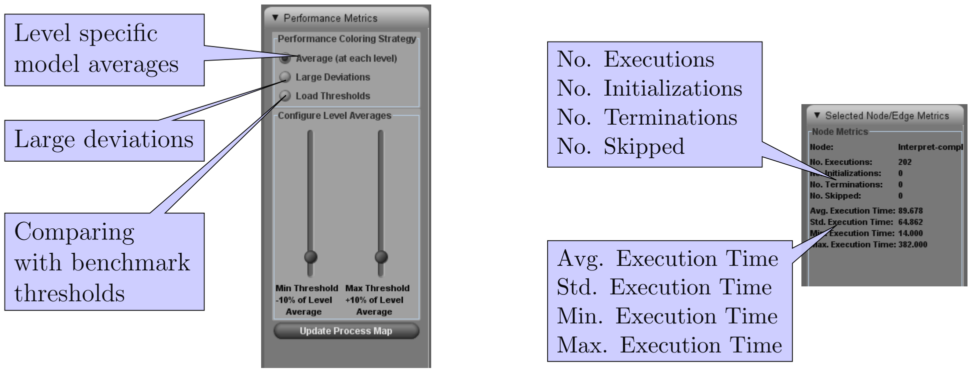

Annotated Mutable Fuzzy Graph
Create new...
PIP
Zoom
Export
Annotated Fuzzy map
Configuring the performance coloring strategy
Inspecting the performance measures of an edge/node
Global performance measures of the event log
Performance Metrics
Performance Coloring Strategy
Average (at each level)
Large Deviations
Load Thresholds
Configure Level Averages
Min Threshold -10% of Level Average
Max Threshold +10% of Level Average
Update Process Map
Selected Node/Edge Metrics
Summary KPI

Figure 10.14: Visualization of the Fuzzy map (hierarchical Fuzzy model) annotated with performance measures.



```
Node:                  Interpret-compl
No. Executions:        202
No. Initializations:   0
No. Terminations:      0
No. Skipped:           0
Avg. Execution Time:   89.678
Std. Execution Time:   64.862
Min. Execution Time:   14.000
Max. Execution Time:   382.000
```

(a) Performance Coloring Configuration (b) Selected Edge/Node Metrics

Figure 10.15: Configuration of performance coloring strategy and exploration of performance measures of any edge/node.

10.6 Trace Alignment with Guide Tree

The Trace Alignment with Guide Tree plug-in caters to the alignment of traces as explained in Chapter 8. Figure 10.16 depicts the framework (cf. Section 8.3) for trace alignment annotated with the plug-ins that realize each step in the framework. Some trivial preprocessing techniques such as the filtering of traces based on their length are available as filter plug-ins in ProM. The construction of guide tree is enabled by the Guide Tree Miner plug-in, which is explained in Section 10.2. The rest of the steps are handled by the Trace Alignment with Guide Tree plug-in. This plug-in takes as input a guide tree generated by the Guide Tree Miner plug-in. Figure 10.17 depicts the visualization of the computed alignments. The alignment visualizer provides rich interactive functionality for the user to explore and gain insights into the process

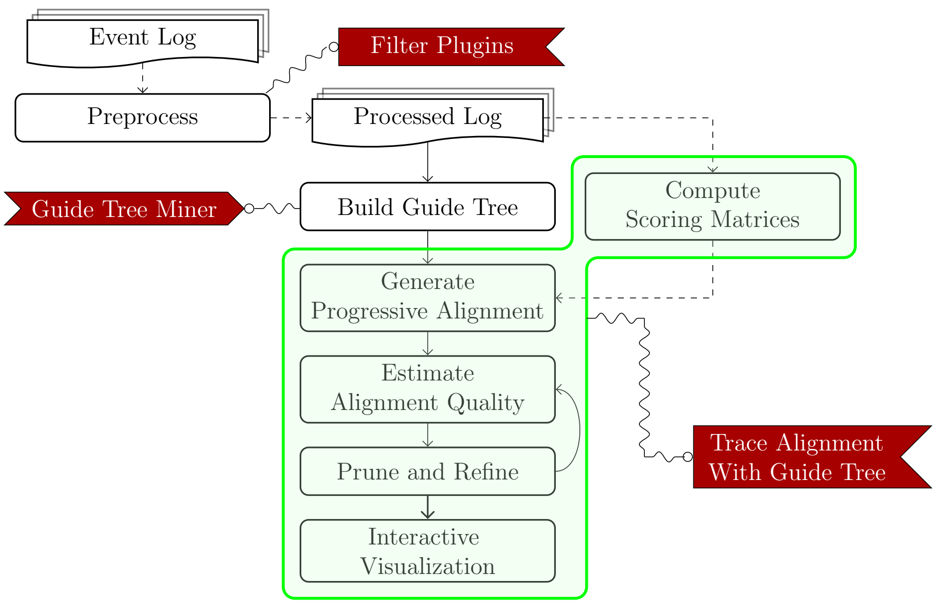


Figure 10.16: ProM plug-ins realizing the framework for trace alignment.

execution. For example, as illustrated in Figure 10.17, the 'View' menu has options for enabling the sorting or filtering of activities in a column of an alignment and other options pertaining to how an alignment is rendered. The analysis menu provides options for refining alignments (based on the concepts presented in Section 8.4) as well as for uncovering the potential concurrent activities in the process. Concurrent activities, if any, are then notified in a separate panel. Furthermore, if the user wishes to manually refine an alignment, there exists a functionality to do so by editing a trace in an alignment as illustrated in Figure 10.17.

Figure 10.18 highlights some of the results of the plug-in. The left panel depicts the process instance identifier (as in the log) and identifiers with a grey background indicate traces that have identical duplicates. The number within parenthesis indicate the number of duplicates. For example there are 7 traces identical to process instance 1018 (corresponding to the trace `jgcflebd` in the event log) while there are no identical traces for the process instance 1127. The top panel depicts a sorting component where the traces involved in the alignment can be sorted based on the activities in a column and the number in the column indicates the priority of sorting. For example, in Figure 10.18, the traces are sorted based on activity `f` with traces having `f` in column 4 having first priority and then with those having `f` in column 7 and finally with those having f in column 11. The bottom panel depicts the information score metric (cf. Section 8.3) for each column as well as a consensus sequence for the alignment.

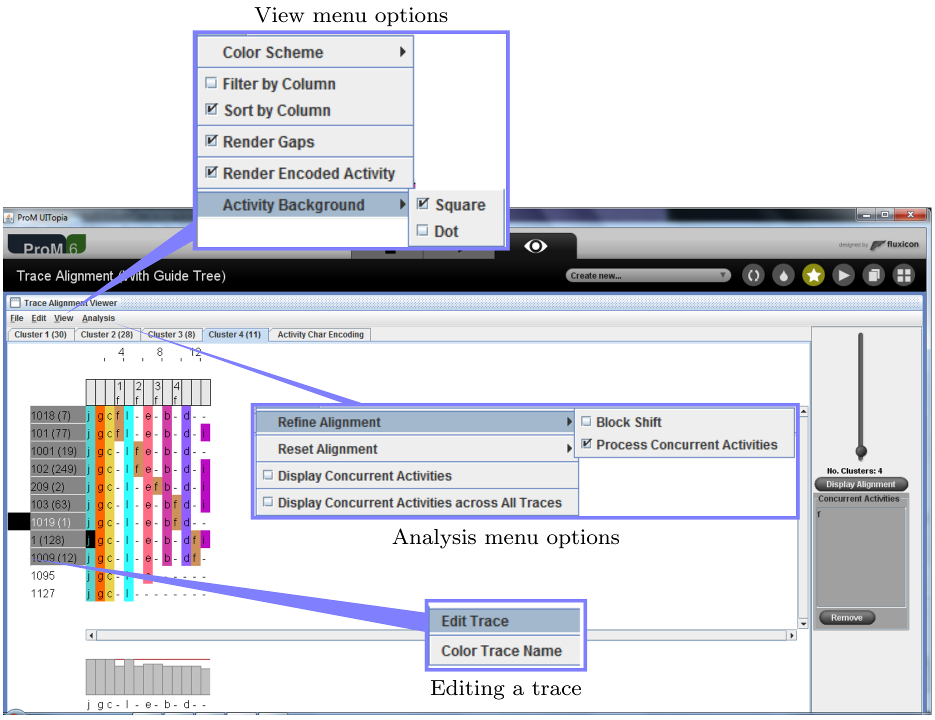


Figure 10.17: Trace alignment visualization with interactive features.

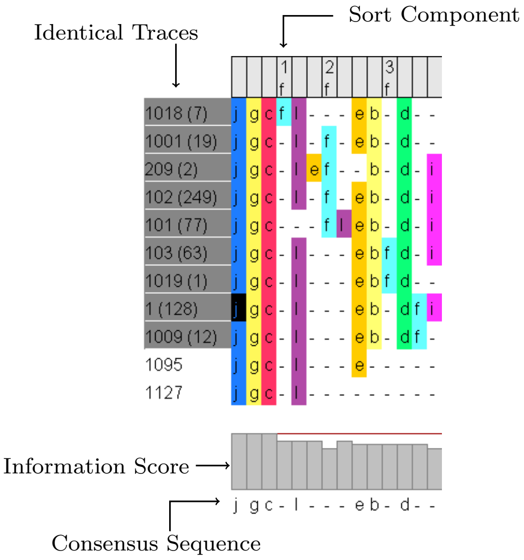


Figure 10.18: Interpreting the various results of trace alignment.

10.7 Signature Discovery

The Signature Discovery plug-in realizes the ideas presented in Chapter 9. Given an event log where the cases are labeled (indicating different classes of behavior), this plug-in serves the purpose of uncovering discriminatory patterns that distinguishes between the different classes of behavior. This plug-in assumes that the label of a case is provided as an attribute value with the key “Class” in the event log. Figure 10.19 depicts the building blocks of the plug-in in line with the framework suggested in Section 9.4 (cf. Figure 9.6).

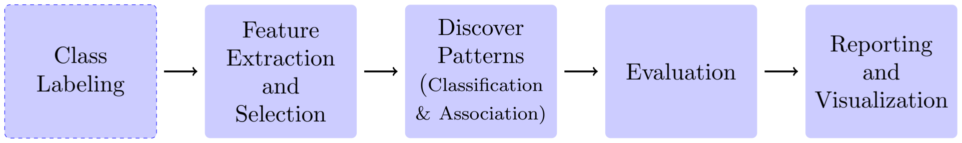
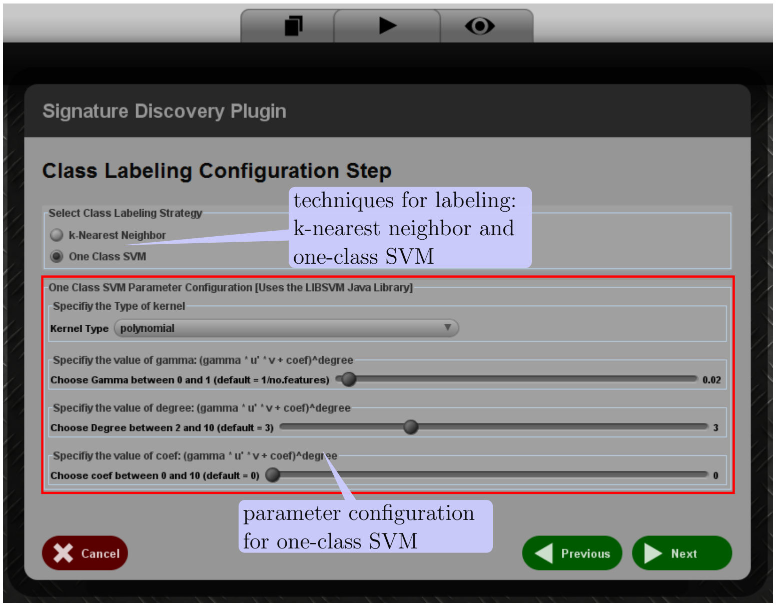

Class Labeling
Feature Extraction and Selection
Discover Patterns (Classification & Association)
Evaluation
Reporting and Visualization

Figure 10.19: Building blocks of the Signature Discovery plug-in.

- *Class Labeling:* the plug-in supports scenarios where some of the cases in the event log are unlabeled. The plug-in supports strategies as discussed in Section 9.4 based on two machine learning algorithms, viz., k-nearest neighbor [46] and one-class SVM (support vector machines) [200], for assigning a label to unlabeled instances. Figure 10.20 depicts the configuration step for class labeling using the one-class SVM technique. We use the libsvm [33] library for support vector machines to support this step. This step also includes the parameter configuration (if any) for the algorithms, e.g., Figure 10.20 depicts the configuration of various parameters for the one-class SVM as supported in libsvm.
- *Feature Extraction and Selection:* the plug-in supports different types of features, which form the basis of the discovered signature patterns. In addition to individual activities, the plug-in supports the various feature sets based on common execution patterns suggested in Section 4.2. Once a feature set or combination of feature sets is chosen, the plug-in transforms each instance in the event log into a vector space defined by the chosen feature set(s). If multiple feature sets are chosen, the union of all the features is considered. Figure 10.21 depicts the configuration step for feature extraction and selection. The plug-in also supports an exploration of all feature spaces and choosing the best feature set. However, this may require long computation times, and therefore not feasible for larger event logs.
- *Pattern Discovery:* Once the features are chosen, the plug-in proceeds to discover the signature patterns using decision trees or association rule mining. The plug-in supports two decision tree learning algorithms ID3 [176] and C4.5 [177] and extends the implementations of these algorithms in the weka library [102, 268]. The extension primarily concerns the prohibition of the discovery of rules with feature constraints that overlap between multiple sessions as discussed in Section 9.4. The plug-in has a custom implementation of mining the

class association rules. Figure 10.22 depicts the learning algorithm configuration step. Figure 10.22 also depicts the parameter configuration step for the chosen algorithm.


Signature Discovery Plugin
techniques for labeling: k-nearest neighbor and one-class SVM
parameter configuration for one-class SVM
Cancel
Previous
Next

Class Labeling Configuration Step

Select Class Labeling Strategy
- [ ] k-Nearest Neighbor
- [✓ One Class SVM]

One Class SVM Parameter Configuration [Uses the LIBSVM Java Library]

Specifiy the Type of kernel
Kernel Type: polynomial

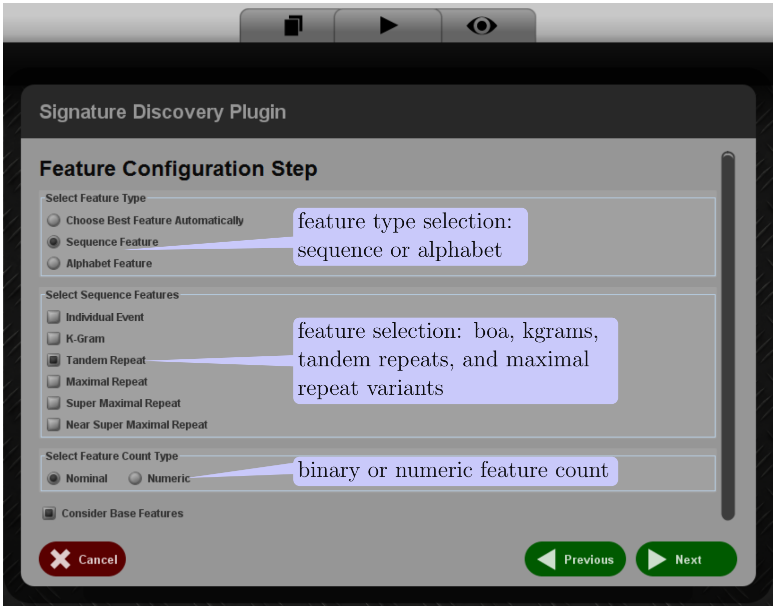
Specifiy the value of gamma: (gamma * u' * v + coef)^degree
Choose Gamma between 0 and 1 (default = 1/no.features): 0.02

Specifiy the value of degree: (gamma * u' * v + coef)^degree
Choose Degree between 2 and 10 (default = 3): 3

Specifiy the value of coef: (gamma * u' * v + coef)^degree
Choose coef between 0 and 10 (default = 0): 0

Figure 10.20: Configuration step for labeling unlabeled instances in an event log. The plug-in supports two algorithms, viz., k-nearest neighbor and one-class SVM for class labeling.

- *Evaluation:* As discussed in Section 9.4, the plug-in supports the evaluation of the discovered signatures using k-folds cross validation. In addition, the plug-in supports the evaluation using the training set and an independent test set. Figure 10.22 depicts the different strategies that can be used for evaluation.
- *Reporting and Visualization:* Figure 10.23 shows the results provided by the plug-in. The plug-in displays the different signatures uncovered for each class of behavior or for chosen classes of behavior. The plug-in also displays the metrics for each of the signatures assessing their quality. The estimated metrics are true (false) positives, true (false) negatives, and derived metrics on these, such as accuracy, sensitivity, specificity, and F1-score [104]. The plug-in also supports the exporting of the signatures and the inspection of instances that satisfy a selected set of signatures.


Signature Discovery Plugin
feature type selection: sequence or alphabet
feature selection: boa, kgrams, tandem repeats, and maximal repeat variants
binary or numeric feature count
Cancel
Previous
Next

Feature Configuration Step

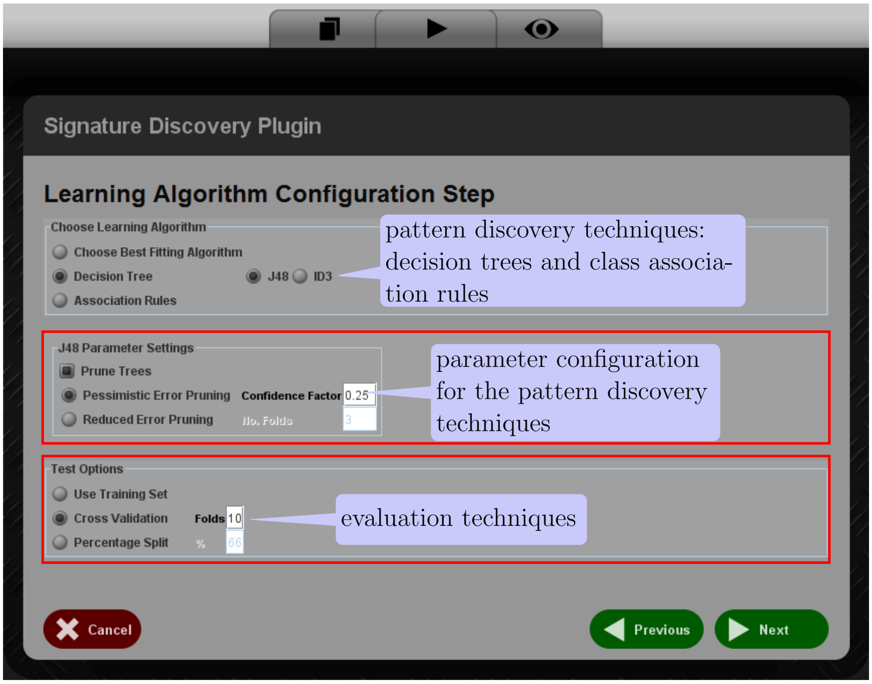
Select Feature Type
- Choose Best Feature Automatically
- [✓ Sequence Feature]
- Alphabet Feature

Select Sequence Features
- Individual Event
- K-Gram
- [✓ Tandem Repeat]
- Maximal Repeat
- Super Maximal Repeat
- Near Super Maximal Repeat

Select Feature Count Type
- [✓ Nominal] Numeric

[✓ Consider Base Features]

Figure 10.21: Configuration step for feature extraction and selection. Different types of features are supported, e.g., sequence and alphabet features: tandem arrays, maximal repeats and variants, and individual events.


Signature Discovery Plugin
pattern discovery techniques: decision trees and class association rules
parameter configuration for the pattern discovery techniques
evaluation techniques
Cancel
Previous
Next

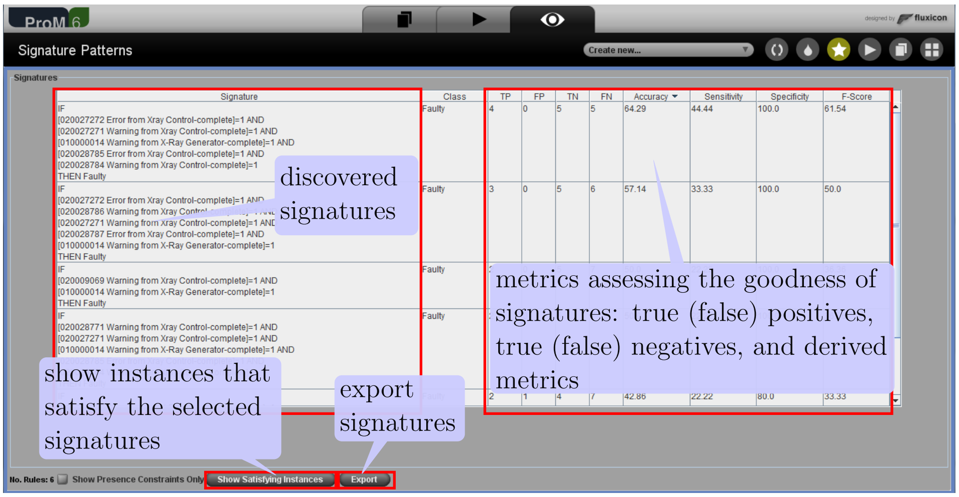
Learning Algorithm Configuration Step

Choose Learning Algorithm
- Choose Best Fitting Algorithm
- [✓ Decision Tree] [✓ J48] ID3
- Association Rules

J48 Parameter Settings
- [✓ Prune Trees]
- [✓ Pessimistic Error Pruning] Confidence Factor 0.25
- Reduced Error Pruning No. Folds 3

Test Options
- Use Training Set
- [✓ Cross Validation] Folds 10
- Percentage Split % 66

Figure 10.22: Configuration step for the learning algorithm for discovering signature patterns. Two classes of algorithms, viz., decision trees and association rule mining are supported.


ProM 6
designed by fluxicon
Signature Patterns
Create new...
Signatures
discovered signatures
metrics assessing the goodness of signatures: true (false) positives, true (false) negatives, and derived metrics
show instances that satisfy the selected signatures
export signatures
No. Rules: 6
Show Presence Constraints Only
Show Satisfying Instances
Export

Signature	Class	TP	FP	TN	FN	Accuracy	Sensitivity	Specificity	F-Score
IF [020027272 Error from Xray Control-complete]=1 AND [020027271 Warning from Xray Control-complete]=1 AND [010000014 Warning from X-Ray Generator-complete]=1 AND [020028785 Error from Xray Control-complete]=1 AND [020028784 Warning from Xray Control-complete]=1 THEN Faulty	Faulty	4	0	5	5	64.29	44.44	100.0	61.54
IF [020027272 Error from Xray Control-complete]=1 AND [020028786 Warning from Xray Control-complete]=1 AND [020027271 Warning from Xray Control-complete]=1 AND [020028787 Error from Xray Control-complete]=1 AND [010000014 Warning from X-Ray Generator-complete]=1 THEN Faulty	Faulty	3	0	5	6	57.14	33.33	100.0	50.0
IF [020009069 Warning from Xray Control-complete]=1 AND [010000014 Warning from X-Ray Generator-complete]=1 THEN Faulty	Faulty	2	0	5	7	50.0	22.22	100.0	36.36
IF [020028771 Warning from Xray Control-complete]=1 AND [020027271 Warning from Xray Control-complete]=1 AND [010000014 Warning from X-Ray Generator-complete]=1 AND [020028785 Error from Xray Control-complete]=1 AND [illegible] THEN Faulty	Faulty	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
IF [illegible]	Faulty	2	1	4	7	42.86	22.22	80.0	33.33

Figure 10.23: Results of the Signature Discovery plug-in. The plug-in estimates different quality metrics for each of the discovered signatures.

Chapter 11
Case Studies

In this chapter, we describe three different case studies where the concepts developed in this thesis have been applied. These case studies have been chosen to illustrate some of the challenges in analyzing large scale event logs and to demonstrate the applicability of the results presented in this thesis. The first case study involves the treatment procedures of patients visiting the gynaecology department in a large academic hospital in the Netherlands (Section 11.1). This case study highlights the significance of dealing with fine-granular events and heterogeneity in event logs. We use the concepts of clustering, hierarchical process discovery, and process diagnostics using trace alignment to analyze the treatment procedures of patients. Our second case study focuses on the handling of concept drifts (Section 11.2). In this case study, we analyze event logs of three different processes from a large Dutch municipality. We employ the techniques proposed in this thesis for change point detection and change localization and demonstrate their capability in correctly identifying process changes. Our third case study has been performed in collaboration with a global leader in professional and consumer healthcare. We analyze the event logs from their high-tech medical systems and analyze the workflow of field service engineers during fault diagnosis (part replacements) (Section 11.3). This case study specifically highlights the challenges in analyzing fine-granular event logs. We use the concepts of abstractions of events and the two-phase approach to process discovery to uncover comprehensible process models. We also demonstrate the annotation of process models with performance information for gaining useful insights in identifying bottlenecks in a process.

Table 11.1 depicts an overview of the three case studies along with some of the characteristics of the processes/event logs and the techniques used in their analysis.

11.1 Patient Treatment Procedures in a Dutch Academic Hospital

In this case study, we investigate the diagnostic and treatment procedures of patients visiting the gynaecology department in the AMC hospital, a large academic hospital located in Amsterdam, The Netherlands. More specifically, it deals with patients, diagnosed with cancer, visiting the oncology clinic in the gynaecology department. Gynaecologic oncology is a specialized field of medicine that focuses on cancers orig-

Table 11.1: Brief description of case studies, their event log/process characteristics, and the techniques used in analysis.

Case Study	**Event Log/ Process Characteristics**	**Techniques**					
		Event Abstractions	Trace Clustering	Concept Drift	Discovering Hierarchical Models	Performance Analysis	Trace Alignment
Analysis of treatment procedures on patients diagnosed for gynaecological cancers in a large academic hospital (AMC)	fine-granular events, heterogenous and long-running cases, complex processes involving interactions between multiple departments		✔		✔		✔
Analysis of process changes in three different processes in a large municipality in the Netherlands	well-structured processes but processes change over time due to various factors; this leads to heterogeneity in event logs			✔			
Analysis of diagnosis procedures followed by field service engineers during part replacements in medical systems	fine-granular events, heterogenous systems, no well-defined notion of a process instance, less structured and flexible processes	✔			✔	✔	

inating in the female reproductive system. It includes cancer of the cervix, fallopian tubes, ovaries, uterus, vagina, and vulva. The International Agency for Research on Cancer indicates that gynaecological cancers accounted for 19% of the 5.1 million estimated new cancer cases, 2.9 million cancer deaths, and 13 million 5-year prevalent cancer cases among women in the world in 2002 [198]. The diagnosis and treatment procedures for patients that are suspected to have gynaecological cancers involve various medical imaging tests, pathology tests, radiotherapies, chemotherapies, and surgical treatments. These procedures are administered in close coordination by a multidisciplinary team of gynaecological oncologists, medical oncologists, pathologists, radiation oncologists, and nurses. This indicates that *the diagnostic and treatment procedures are non-trivial.*

The designated gynaecological cancers workforce (ones specialized in the gynaecological cancers) and available resources (be it labs, medical imaging equipment, and other medical professionals such as radiologists and anesthesiologists) are limited, and therefore precious, not just in the AMC but in any hospital around the globe. This, compounded with an ever increasing demand, presents many challenges for healthcare organizations such as AMC. For example, in workforce planning: how does it affect the coordination of health services? how does it impact on changing models of care? what are the implications for patient outcomes?, etc [126]. Hospitals have to focus on ways to streamline their processes in order to deliver high quality care while at the same time reducing costs [9]. Furthermore, government and health insurance companies impose pressure on hospitals to work in the most efficient way. A better understanding of current practices can go a long way in shaping the treatment and care of women with gynaecological cancers.

AMC is interested in understanding the processes corresponding to patient diagnosis and treatments. More specifically, they are interested in gaining insights into the number and ordering of diagnosis tests performed, regularities (common patterns of execution) and irregularities (violations) in the tests performed, process variations such as variations in their medical practices, variations in logistic scheduling and arrangements, etc. Furthermore, they are interested in obtaining insights on process diagnostics such as the circumstances under which irregularities are observed.

We consider the event log provided for the first *Business Process Intelligence Challenge* (BPIC). The event log corresponds to the diagnosis of gynecological oncology patients once they are referred to the AMC hospital for treatment. Since patients are referred (by other hospitals), some medical tests could have already been performed at the referring hospital and that a patient may already be (partly) diagnosed. The diagnostic trajectory performed at AMC aims at further diagnosis of a patient and deciding on the strategy that needs to be followed for treatment of a patient. Additional steps may be performed requesting the referring hospital to send material to AMC so that all necessary details about a patient are available to the medical professionals involved. In total, the diagnostic trajectory of the gynecological oncology healthcare process is significant and consists of over 300 tasks [146]. This shows that we are dealing with a *complex* process.

The event log contains 1143 cases and $150,291$ events referring to 624 activities where the activities pertain to the treatment procedures that are being administered on patients diagnosed with gynaecological cancers during the period between March 2005 and March 2008. A naive attempt at analyzing this event log using existing process mining techniques is bound to provide results that are incomprehensible and unsatisfactory, e.g., control-flow analysis of the log generates a workflow that is a graph containing a few hundred nodes (each node corresponding to an activity) and edges connecting the nodes based on their dependency. Figure 11.1 depicts the heuristic net mined on the event log. We can see that the mined model is spaghetti-like and incomprehensible. One cannot even explore the model, leave alone getting any meaningful insights. This log exhibits some of the characteristics of large scale event logs, e.g., the activities in the event log are fine-grained (this event log contains 624 different activities), the event log is heterogenous (it contains a heterogenous mix of patients diagnosed for cancer (at different stages of malignancy) pertaining to the

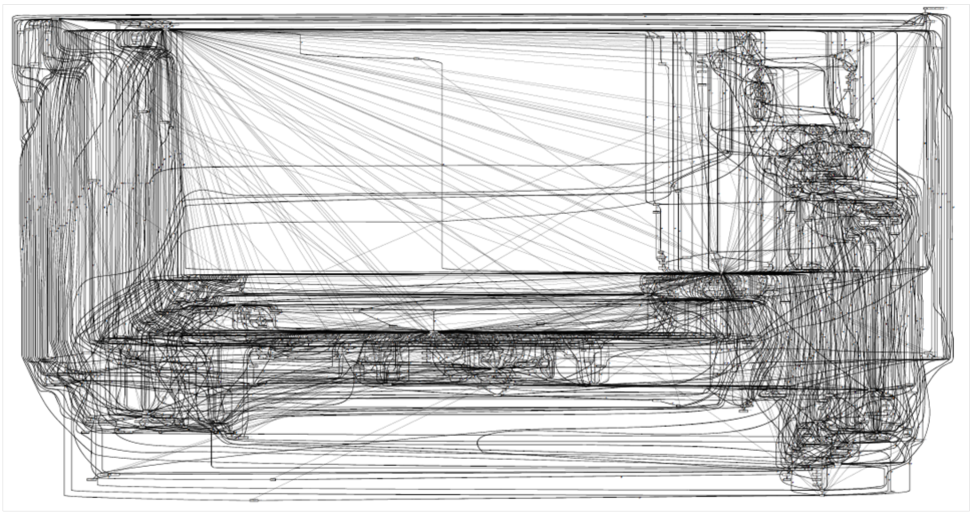


Figure 11.1: Process model discovered using the heuristic miner [264] on an event log describing the diagnosis and treatment of 1143 patients of the Gynaecology department in a Dutch hospital.

various organs such as the cervix, vulva, uterus and ovary), etc. The spaghettiness of the model in Figure 11.1 is largely attributed to these factors. As has been advocated in this thesis, *preprocessing of the log* leading to event log simplification is an *essential* step in gaining meaningful insights.

Figure 11.2 provides a high-level view of our approach used to analyze this log. We focus on two perspectives of process mining, viz., control-flow and process diagnostics. More specifically, from the control-flow perspective, we attempt at extracting the workflow of treatment procedures catered to patients, and from the process diagnostics perspective, we attempt at identifying regularities and irregularities/violations in treatment patterns across patients. As mentioned earlier, AMC is interested in both of these aspects.

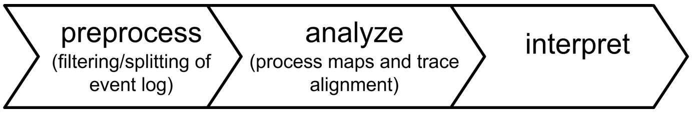

preprocess
(filtering/splitting of event log)
analyze
(process maps and trace alignment)
interpret

Figure 11.2: Overview of the approach followed.

11.1.1 Preprocessing

We first filter (remove) all administrative activities from the event log. These activities are considered to be irrelevant for the analysis of patient treatment procedures. More specifically, we removed the following 12 activities: 190101 reg.toesl above. A101, 190204 A204 Class 3a, 190205 Class 3b A205, administrative fee - the first pol, clinical card - internal medicine, clinical map - anesthesia, inwend.geneesk. aanv.kaart cost-Out, inwend.geneesk. Out-year card costs, inwend.geneesk. short-out card cost, verlosk.-gynaec. Out-year card costs, verlosk.-gynaec. short-out card cost, verlosk.-gynaec. suppl. map-out costs. After this step, the event log contains 1143 cases and 131,096 events distributed over 612 activities. The event log contains rich information stored as attributes both at the event level and at the case level. We exploit this information and propose a few perspectives for preprocessing.

Diagnosis Perspective

Each case contains a few attributes that provide information on the illness the patient is diagnosed with. The event log captures treatment procedures pertaining to different types of cancer in 11 different diagnosis values. For example, Table 11.2 depicts that the diagnosis value M11 corresponds to the cancer of the vulvar region, which in turn is classified into different types and stages such as SCC (squamous cell carcinoma) of stages I, II, III1, III2, IVa, and IVb.

Table 11.2: Description of different types of diagnosis associated with the diagnosis value M11

Diagnosis Value	Diagnosis Description
M11	Pertains to the cancer of the vulva. Describes information on cases diagnosed with • squamous cell carcinoma (stages I, II, III1, III2, IVa, and IVb) • malignant neoplasms and melanoma • basal cell carcinoma • borderline malignancy

Table A.1 in Appendix A depicts the complete list of diagnosis values and the types of cancer that they refer to. The different types of cancer captured in the event log is in accordance with the FIGO (International Federation of Gynaecology and Obstetrics) staging of cancers pertaining to gynaecological cancers [12]. Figure 11.3 depicts the Venn diagram of the various diagnosis values and the regions to which they are associated.

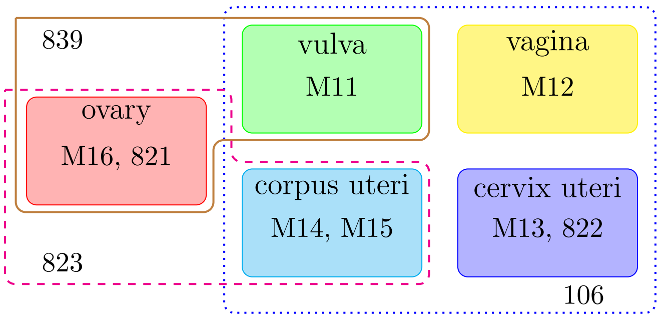


Figure 11.3: Diagnosis values and the regions to which they are associated.

Due to the different types of cancer and differences among patients, the cases found in the event log can be characterized as *heterogeneous*. This heterogeneity adds to the complexity of analysis and the results (see Figure 11.1) are often incomprehensible. As discussed in Chapter 4, process mining results can be improved by partitioning an event log into subsets of homogenous cases and analyzing these subsets independently. We now propose a few means of segregating homogenous cases based on the diagnosis perspective.

- **Individual Diagnosis Value:** One can filter the event log based on a particular value for any of the diagnosis, e.g., consider cases where the diagnosis value = M14, consider cases where the diagnosis is 'squamous cell carcinoma stage Ib'. One can also use a combination of both the diagnosis value and the specific diagnosis to select cases for analysis, e.g., cases where the diagnosis value = M11 and the diagnosis is 'basal cell carcinoma'.
- **Diagnosis Value Combination:** Since cases may contain multiple diagnosis values, one can alternatively look at the combination of values. For example, a case can have {M13, 822, 106} as the set of diagnosis values. If the event log is preprocessed for all such combinations, we notice certain relationships between the values in the diagnosis among the cases. For example, there are distinct cases where the combinations {M13}, {M13, 822}, and {M13, 822, 106} exist. We can clearly see a subsumption property between the values: $\{M13\} \subset \{M13, 822\} \subset \{M13, 822, 106\}$. The set of treatment procedures applied on patients with diagnosis value combination {M13, 822} typically includes the procedures applied to patients with diagnosis value {M13}. We can capture the relationships between the diagnosis value combinations manifested in the event log using a Hasse diagram by considering a partial ordering (with subsumption as the cover relation) on the value combinations. In the event log, there are a total of 38 distinct diagnosis value combinations. Table A.2 in Appendix A depicts the distribution of cases over the different diagnosis value combinations. Figure 11.4 depicts the Hasse diagram corresponding to the value combinations involving M13. The figure also depicts the number of cases in the event log for each diagnosis value combination, e.g., there are 57 cases with the combination {M13, 106}.

The nodes in the Hasse diagram form the basis for segregation of homogenous cases for analysis. Two types of node selection mechanisms can be adopted.

- *Single-node mode:* In this mode, one can consider all cases where the diagnosis value combination of the selected node is manifested, e.g., choosing the node {M13, 106} implies considering only those cases pertaining to patients who have been diagnosed with both M13 and 106.
- *Sub-graph mode:* In this mode, multiple nodes can be selected. This implies the union of all cases where the diagnosis value combinations of the selected nodes are manifested are considered to be homogenous, e.g., selecting all the nodes subsumed under the maximal element {M13, 106, 822} implies considering all

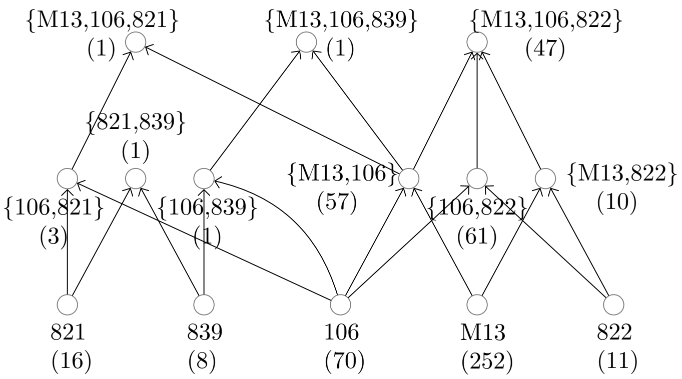


Figure 11.4: Hasse diagram corresponding to the diagnosis value combinations involving M13 and related diagnosis. The number in parenthesis () signify the number of cases in the event log with that value combination.

the cases pertaining to patients who have been diagnosed with either {M13, 106, 822} or {M13, 106} or {M13, 822} or {106, 822} or {M13} or {106} or {822} as homogenous.

Treatment Perspective

The event log also contains information corresponding to the treatment administered on the patients. Each case may contain one or more treatments specified. 1131 of the 1143 cases in the event log have at least one treatment value specified. There are a total of 46 distinct values for the treatments and 236 distinct treatment value combinations in the event log. A vast majority of treatment value combinations are unique combinations. One can also use the treatment values as a basis for defining homogeneity of cases with the method illustrated using Hasse diagram shown in Figure 11.4. However, now treatment values instead of diagnosis values are used.

Time Perspective

The event log contains cases (pertaining to patients) during the period between March 2005 and March 2008. We define the span period (or duration) of a case to be the time difference between the last and first events of the case. Figure 11.5 depicts the histogram of the span period of the cases. It can be seen that the cases typically run over a long period of time (the average span period is 386 days and the standard deviation is 338 days). Since each case in the event log corresponds to a patient, one can interpret the long durations to be the time under which the patient is being treated. During this period, the patient could have visited or consulted the hospital several times, with a plausibility of these visits having to do with multiple problems. Therefore, it is rather unusual to consider all the events in a case as belonging to a single process instance for analysis. This calls for the definition of appropriate notions of process instances, i.e., we will split cases into smaller more representative cases.

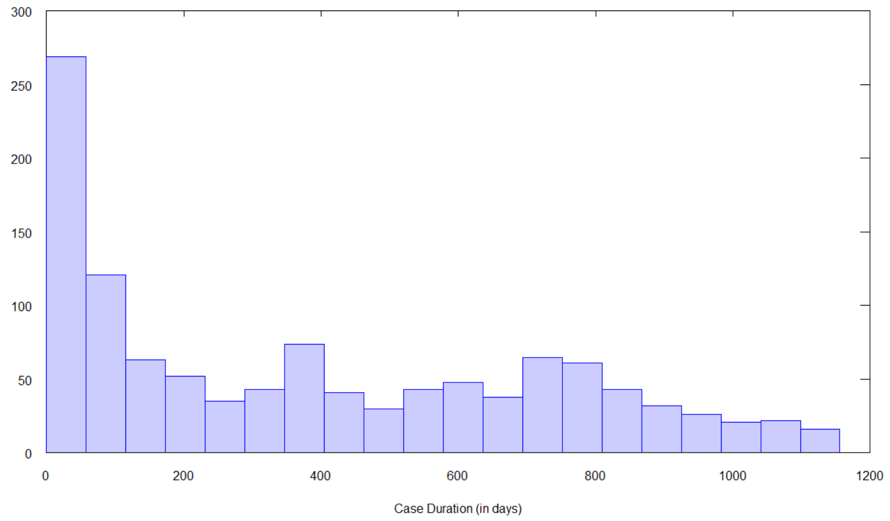


Figure 11.5: Histogram depicting the span period of the cases.

A closer inspection of the log reveals that the activities in a case happen in bursts. Figure 11.6 depicts a typical scenario of activity bursts in cases. Bursts signify periods of consultation/treatment and idle periods denote recuperation, e.g., a patient undergoing radiotherapy visits a hospital at regular intervals and there are recommended time intervals between successive applications of radiotherapy or between radiotherapy and surgery. We exploit this characteristic and define a process instance to capture a burst of activities. One can use a parameter, say δ days, to demarcate the boundaries between process instances. Two events or event sequences with a time period between them greater than δ fall under two process instances as depicted in Figure 11.7.

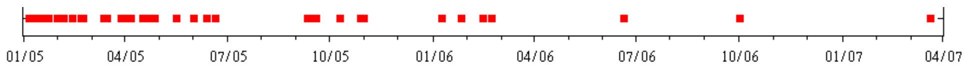


Figure 11.6: The cases in the event log contain bursts of activities at certain points during the span period of the case. The x-axis denotes time with the format month/year. A dot can represent a multitude of events due to the coarse-grained scale used in x-axis.

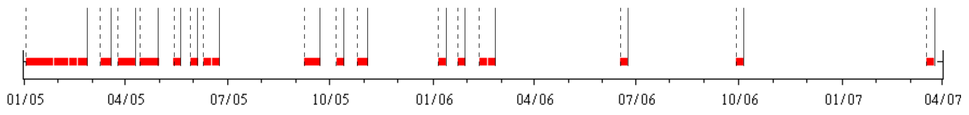


Figure 11.7: Defining process instances within a case based on activity bursts and idle periods. The dotted lines indicate the start of a process instance and the immediate following solid line indicates the end of the process instance.

Using the Organizational Perspective to Derive Artifacts

Each event in the event log contains an attribute 'org:group' that captures the department/lab where the activity corresponding to the event was performed. There are 43 distinct org:group values (departments/labs) in the event log with one being 'unknown'. The process instances (bursts of events) defined above exhibit certain regularity with respect to the organization group. The regularity is often manifested as a related set of diagnosis tests in the form of a continuous series of activities, e.g., different diagnosis blood tests prescribed for a patient in the lab, as illustrated in Figure 11.8. This can be associated to the concept of *artifacts* [161] in business processes, i.e., process instances may be decomposed into more fine-grained instances. In the original event log, we have events at the level of a blood test interleaved with the events at the diagnosis or treatment level. We can try to decompose them and create separate instances, e.g., instances capturing only events related to blood tests.

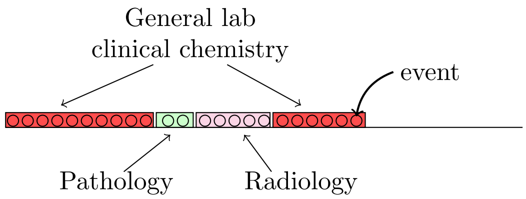


Figure 11.8: Regularities in the form of series of activities (pertaining to related diagnosis tests) performed in a lab/department.

Exploiting this notion, we propose the transformation of the original log into an *abstraction log* where the activities correspond to the organization names. Each continuous sequence of one or more events pertaining to the same organization in the process instance of the original log is replaced by a single event with the organization name as its activity. At the same time, we create one sub-log for each organization whose process instances correspond to the replaced sequence of events. The transformation process is illustrated by Figure 11.9. The process instance in Figure 11.8 is transformed into the process instance GPRG... where G, P, and R are used as short-cuts for the organizations (departments), general lab clinical chemistry (G), pathology (P), and radiology (R) respectively. At the same time, we create three sub-logs, one for each of G, P, and R. The process instances in these sub-logs correspond to the sequence of events replaced in the original process instance.

Urgent and Non-urgent Cases

The event log contains certain activities that are classified as urgent. Ordinary counterparts to such activities also exist. For example, the activities haemoglobin photoelectric and haemoglobin photoelectric-urgent both exist (this activity corresponds to the estimation of haemoglobin using a photoelectric calorimeter). Similarly, the activities platelet count and platelet count-urgent both exist. There are a total of 28 urgent activities in the event log. This indicates that certain cases (patients) are considered as emergency cases and are treated in an expedited manner. These

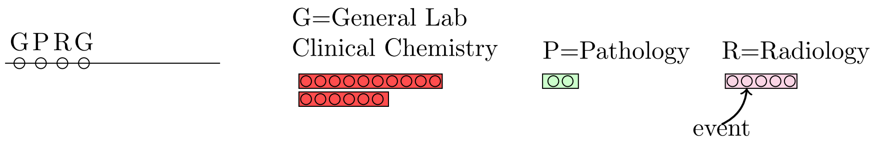


Figure 11.9: Transformation of the original log into an abstraction log using the notion of artifacts on the organizational perspective. The activities in the abstraction log correspond to the organization names. Also, one sub-log is created for each organization.

urgent activities are mostly executed during and around the time period of surgery on patients. When a patient profusely bleeds during/post surgery, the doctors constantly monitor the biochemical profiles of the patients and these tests need to be done in an expedited manner. We can exploit this information in order to segregate homogenous cases. We can partition a given log, say a log containing cases of patients with diagnosis code combination as {M11}, into two categories: urgent and non-urgent cases. Urgent cases are those cases where at least one activity of type urgent is manifested.

We use the perspectives defined above either in isolation or in combinations as a preprocessing step to segregate homogenous cases based on the focus of analysis. We will show some examples in the next two subsections.

11.1.2 Control-flow Analysis

In this section, we analyze the control-flow perspective of patient flows across the departments.

Workflow of Treatment Procedures on Patients Diagnosed for M11

The event log (after removing the administrative activities) is subjected to the following pre-processing steps.

- Select the cases whose diagnosis value combination is just {M11} (patients who have been diagnosed with vulvar cancer). There are a total of 162 cases in the event log satisfying this criterion. This filtered event log contains 8781 events distributed over 199 activities.
- Segregate urgent and non-urgent cases from the filtered log obtained in the previous step. Of the 162 cases, 137 cases are non-urgent cases containing 4820 events referring to 135 activities while 25 cases are urgent cases containing 3961 events distributed over 165 activities. Let us call these two logs the *non-urgent cases log* and *urgent cases log*. It is interesting to note that though the number of urgent cases is just 25 (15%), it contains almost half (45%) of the total number of events.
- Transform the two logs (i.e., the urgent cases log and the non-urgent cases log) based on the notion of artifacts over the organizational perspective. Selecting this perspective includes the selection of time perspective for the definition of process instances. These process instances are reflected in the sub-logs. The

abstraction log for the non-urgent cases contains 136 cases[1] and 1527 events distributed over 21 activities (corresponding to the organization names). In addition, 21 sub-logs are created, one for each abstract activity (organization name). The abstraction log for the urgent cases contains 25 cases and 1061 events distributed over 18 activities. In addition, 18 sub-logs are created, one for each abstract activity (organization name).

Figure 11.10 depicts the process model based on the abstraction log pertaining to the non-urgent cases discovered using the Fuzzy Map miner. This corresponds to the patient flows across the various labs/departments. All the nodes in the workflow are colored in blue (signifying that they are abstract nodes). Abstract nodes can be seamlessly zoomed in to see the subprocesses underneath it. Figure 11.11(a) depicts the subprocess pertaining to the activities performed on the patients in the pathology department. The pathology department performs histopathological studies on the tissues of patients. Some of the primary activities include identifying the compartment for inspection, resection of tissues (small and big) and performing biopsies on them. Figure 11.11(b) depicts the subprocess pertaining to the activities performed on patients in the radiology department. The radiology department takes image scans of regions. Different modalities of scanning are supported, e.g., MRI, CT, ultrasound, etc. The scans are primarily performed on the pelvis, thorax, abdomen, and allied bones. A CT chest scan is also normally performed. There is one exceptional case for whom additional dental, brain, knee and leg veins scans were performed (the highlighted region in the figure) at different instances in time.

Figure 11.12 depicts the subprocess pertaining to the activities of the 'General Lab Clinical Chemistry'. This lab is concerned with various diagnosis tests on blood and urine. A few classes of tests are highlighted in the figure. For example, tests that assess the levels of creatinine, sodium, phosphate, bicarbonate, etc., are performed. As another example, tests are performed to assess the blood group, estimate Rh factor, the white blood cell counts (leukocyte), red blood cell counts (haematocrit etc.), platelet counts, etc. Sediment and urine analysis tests are also performed on patients. Patients diagnosed with M11 (vulvar cancer) are subjected to the CEA (carcinoembryonic antigen) tumor marker test followed by the cancer antigen tests, CA 125 and CA 19.9. Figure 11.13 depicts the region corresponding to the blood count tests in Figure 11.12.

In a similar fashion, the procedures followed in other departments/labs can be analyzed.

Figure 11.14 depicts the workflow of the abstraction log pertaining to the urgent cases discovered using the enhanced Fuzzy Miner. Figure 11.15(a) depicts the subprocess pertaining to the activities performed on the patients in the pathology department while Figure 11.15(b) depicts the subprocess pertaining to the activities performed on the patients in the radiology department. The activities performed are more or

[1]This log contains one case less than the non-urgent cases log because one case in the non-urgent cases log has certain events without the organizational group attribute/value. We ignored this particular case.

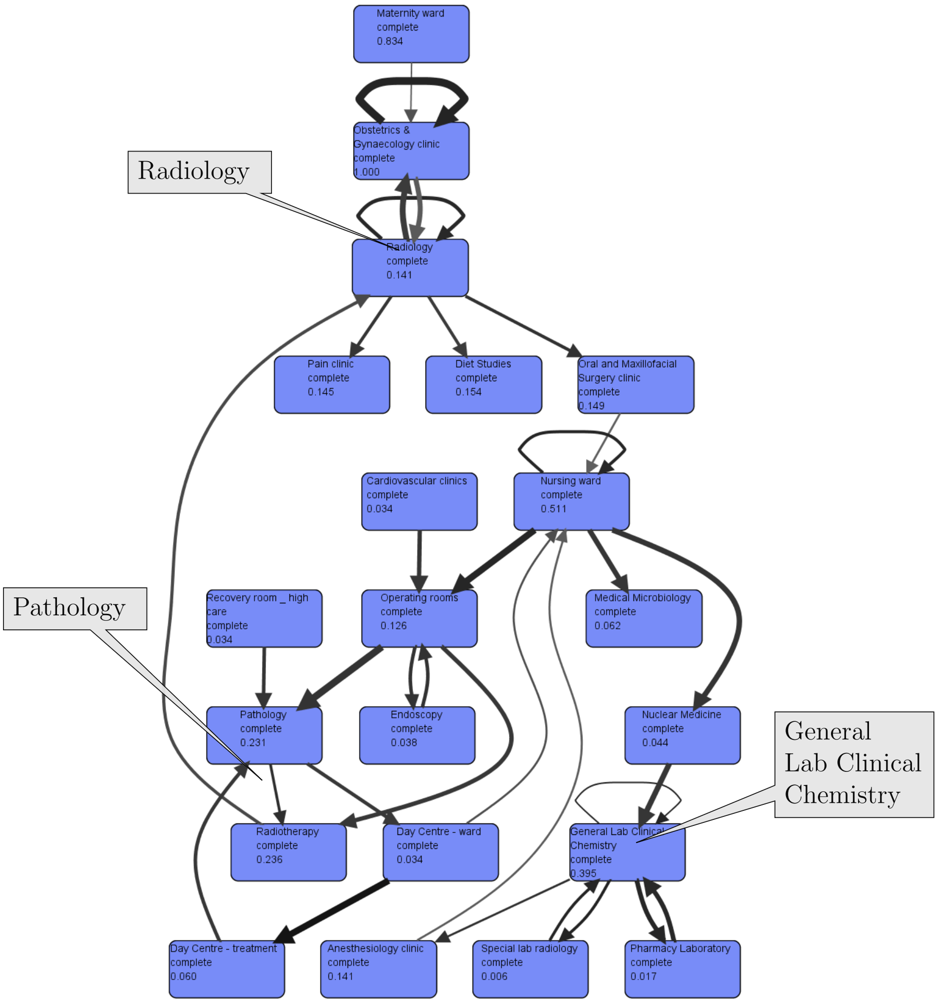
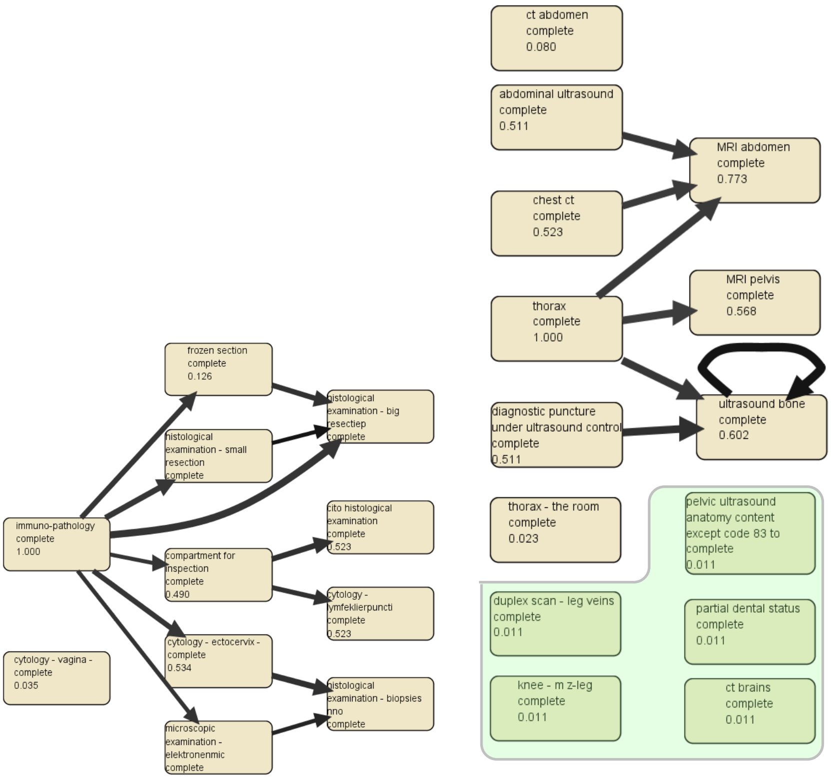


Figure 11.10: The process model depicting the flow of patients across the different labs/departments in the non-urgent cases diagnosed with M11.

less the same as that of the non-urgent cases except that for some urgent cases, 'aspiration cytology' and 'cytology vulva' are performed in the pathology department. The CT scan of the retroperitoneal region was performed on some of the urgent cases while none of the non-urgent cases required this. While only a MRI scan of the pelvis region was performed for the non-urgent cases, some urgent cases had a CT scan of the pelvis in addition to a MRI scan.

The primary difference between the treatment/diagnosis procedures followed between the non-urgent and urgent cases emanate in the general lab clinical chemistry subprocess. As mentioned earlier, 'urgent' variants of the activities (lab tests) are followed in the urgent cases. Figure 11.16 depicts the subprocess of the general lab


immuno-pathology complete 1.000
frozen section complete 0.126
histological examination - small resection complete
histological examination - big resectiep complete
cito histological examination complete 0.523
compartment for inspection complete 0.490
cytology - lymfeklierpuncti complete 0.523
cytology - ectocervix - complete 0.534
histological examination - biopsies nno complete
microscopic examination - elektronenmic complete
cytology - vagina - complete 0.035
ct abdomen complete 0.080
abdominal ultrasound complete 0.511
MRI abdomen complete 0.773
chest ct complete 0.523
thorax complete 1.000
MRI pelvis complete 0.568
diagnostic puncture under ultrasound control complete 0.511
ultrasound bone complete 0.602
thorax - the room complete 0.023
pelvic ultrasound anatomy content except code 83 to complete 0.011
duplex scan - leg veins complete 0.011
partial dental status complete 0.011
knee - m z-leg complete 0.011
ct brains complete 0.011

(a) Pathology (b) Radiology

Figure 11.11: The subprocesses pertaining to the activities performed on the patients in the pathology and radiology departments for the non-urgent cases diagnosed with M11.

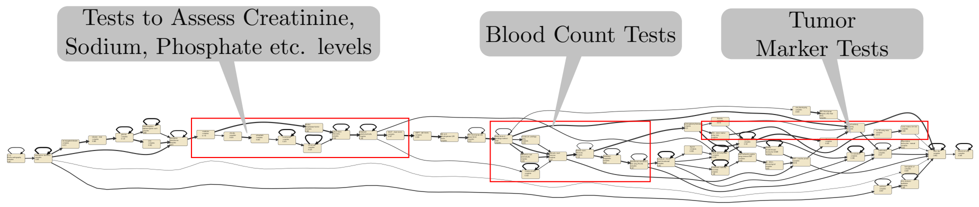


Figure 11.12: The subprocess pertaining to the activities performed on the patients in the general lab clinical chemistry for the non-urgent cases diagnosed with M11.

clinical chemistry. The highlighted regions in the figure indicate the series of tests involving the urgent variants of the activities. The bottom half of the process is almost like that of the non-urgent cases. It is important to note that during the lifetime of a case, the lab tests can be performed multiple times and at certain instances these tests were needed to be done in an expedited manner and at other instances the normal flow was followed. That is the reason why we see both the normal and

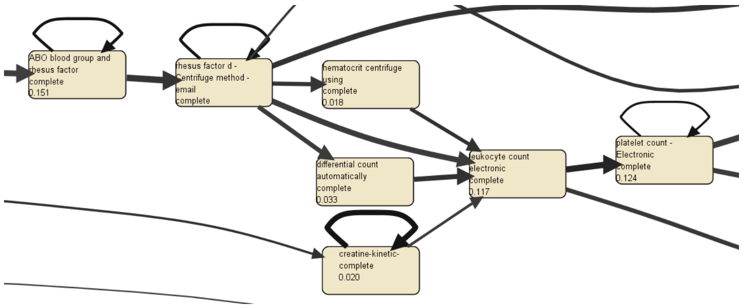


Figure 11.13: A portion of the general lab clinical chemistry subprocess pertaining to the estimation of blood counts for non-urgent cases diagnosed with M11.

urgent variants of the activities.

We have also analyzed the workflow of cases diagnosed with other code values and value combinations. In all the scenarios, we were able to mine meaningful process models. Although at first sight the log seems to be too complex to provide any understandable process models, our systematic approach helped us to discover models that are rather simple and sequential. Furthermore, this event log highlights the influence of heterogeneity in event logs and the significance of partitioning them into homogenous subsets for analysis.

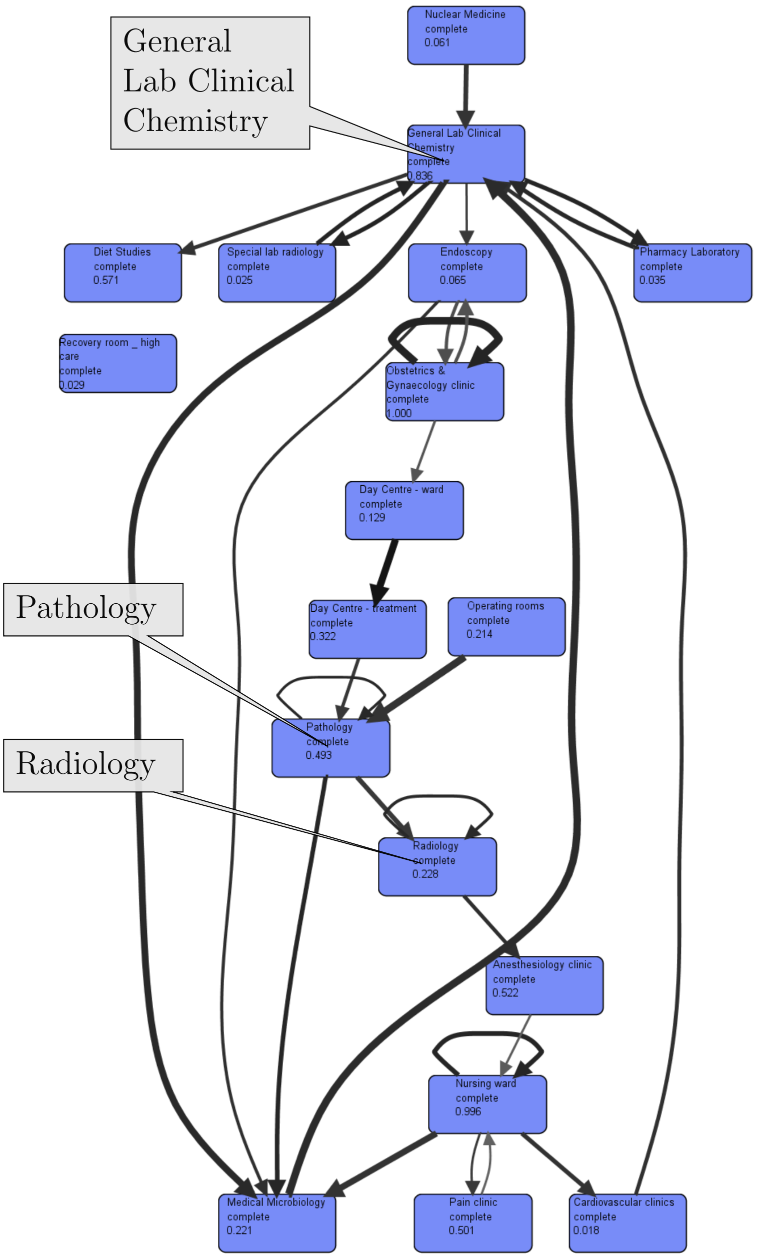
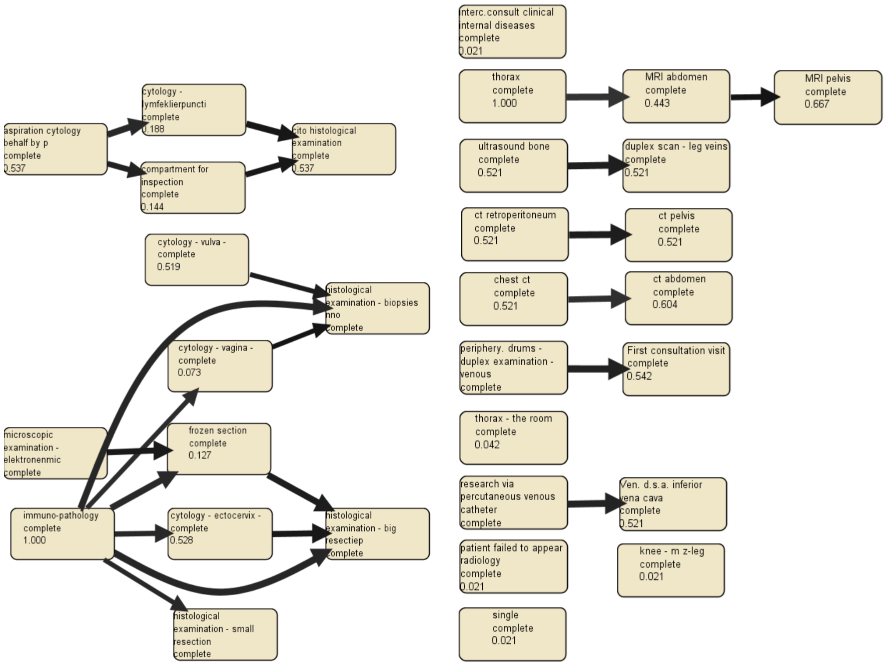

General Lab Clinical Chemistry
Nuclear Medicine complete 0.061
General Lab Clinical Chemistry complete 0.836
Diet Studies complete 0.571
Special lab radiology complete 0.025
Endoscopy complete 0.065
Pharmacy Laboratory complete 0.035
Recovery room _ high care complete 0.029
Obstetrics & Gynaecology clinic complete 1.000
Day Centre - ward complete 0.129
Pathology
Day Centre - treatment complete 0.322
Operating rooms complete 0.214
Radiology
Pathology complete 0.493
Radiology complete 0.228
Anesthesiology clinic complete 0.522
Nursing ward complete 0.996
Medical Microbiology complete 0.221
Pain clinic complete 0.501
Cardiovascular clinics complete 0.018

Figure 11.14: The process model depicting the flow of patients across the different labs/departments in the urgent cases diagnosed with M11.


aspiration cytology behalf by p complete 0.537
cytology - lymfeklierpuncti complete 0.188
compartment for inspection complete 0.144
cito histological examination complete 0.537
cytology - vulva - complete 0.519
histological examination - biopsies nno complete
cytology - vagina - complete 0.073
microscopic examination - elektronenmic complete
frozen section complete 0.127
immuno-pathology complete 1.000
cytology - ectocervix - complete 0.528
histological examination - big resectiep complete
histological examination - small resection complete
interc.consult clinical internal diseases complete 0.021
thorax complete 1.000
MRI abdomen complete 0.443
MRI pelvis complete 0.667
ultrasound bone complete 0.521
duplex scan - leg veins complete 0.521
ct retroperitoneum complete 0.521
ct pelvis complete 0.521
chest ct complete 0.521
ct abdomen complete 0.604
periphery. drums - duplex examination - venous complete
First consultation visit complete 0.542
thorax - the room complete 0.042
research via percutaneous venous catheter complete
Ven. d.s.a. inferior vena cava complete 0.521
patient failed to appear radiology complete 0.021
knee - m z-leg complete 0.021
single complete 0.021

(a) Pathology (b) Radiology

Figure 11.15: The subprocesses pertaining to the activities performed on the patients in the pathology and radiology departments for the urgent cases diagnosed with M11.

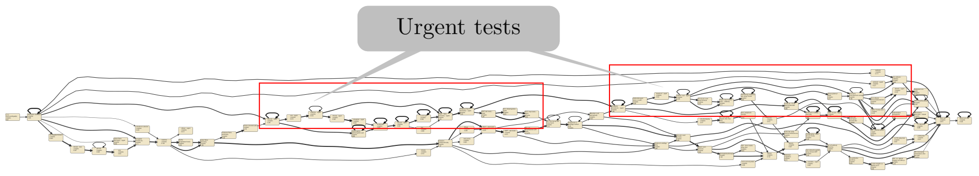


Figure 11.16: The subprocess pertaining to the activities performed on the patients in the general lab clinical chemistry for the urgent cases diagnosed with M11.

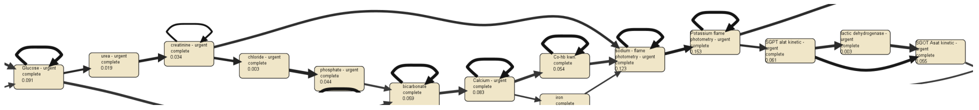

Glucose - urgent complete 0.091
urea - urgent complete 0.019
creatinine - urgent complete 0.034
chloride - urgent complete 0.003
phosphate - urgent complete 0.044
bicarbonate complete 0.059
Calcium - urgent complete 0.083
Co-hb kwt. complete 0.054
iron complete
sodium - flame photometry - urgent complete 0.123
Potassium flame photometry - urgent complete 0.153
SGPT alat kinetic - urgent complete 0.061
lactic dehydrogenase - urgent complete 0.003
SGOT Asat kinetic - urgent complete 0.055

Figure 11.17: A portion of the general lab clinical chemistry subprocess involving the urgent activities pertaining to the estimation of creatinine, sodium, phosphate, etc. levels.

11.1.3 Process Diagnostics using Trace Alignment

We now analyze the AMC log using trace alignment. As mentioned in Chapter 8, trace alignment can be used to explore the process in the early stages of analysis and to answer specific questions in later stages of analysis, e.g., are there common patterns of execution, are there any anomalies, are there any distinguishing aspects with respect to the treatment procedures followed among cases, etc.

We will discuss the application of trace alignment and infer insights by using the cases diagnosed for cervical cancer of the uteri (diagnosis value M13). The raw event log is subjected to the following pre-processing steps

- Select the cases whose diagnosis value combination is the singleton set {M13}. There are a total of 252 cases in the event log satisfying this criteria. This filtered event log contains $12,249$ events distributed over 263 activities.
- From the filtered log obtained in the previous step, select the cases that have been subjected to a treatment code combination of {803}. There are 23 cases satisfying this criteria. This filtered event log contains 2791 events distributed over 129 activities. Though only 9% of the cases diagnosed with {M13} are treated with the treatment code {803}, nearly 23% of the events happen in these 9% of the cases.
- Segregate urgent and non-urgent cases from the filtered log obtained in the previous step. Of the 23 cases, 15 cases are non-urgent cases containing 1647 events referring to 106 activities while 8 cases are urgent cases containing 1144 events distributed over 88 activities. Let us call these two logs the *non-urgent cases log* and *urgent cases log.*

Figure 11.18 depicts the initial portion[2] of the aligned traces pertaining to the cases in the non-urgent event log. The event log is first encoded into traces where each trace is the sequence of activities corresponding to a case. The activities are represented in an encoded form in a trace with each activity encoded using two characters (e.g., `h9`, `a2`, `c4`, etc.). We can clearly see some common patterns of execution and exceptional/rare behavior in the alignment. For example, from the alignment, we can see that

- all the cases with the exception of one (trace 00001023) have the activity sequence `e3c2` corresponding to the estimation of ABO blood group and Rh factor (`e3`) and Rh factor using centrifuge method (`c2`) respectively
- only one of the cases (trace 000000928) required the execution of the activity `e7` corresponding to cephalin time-coagulation test
- only in three cases was the activity `a0` (CEA - tumor marker using meia) performed

[2]The entire alignment is of length 378 and is not shown due to legibility issues. The entire alignment can be provided upon request.

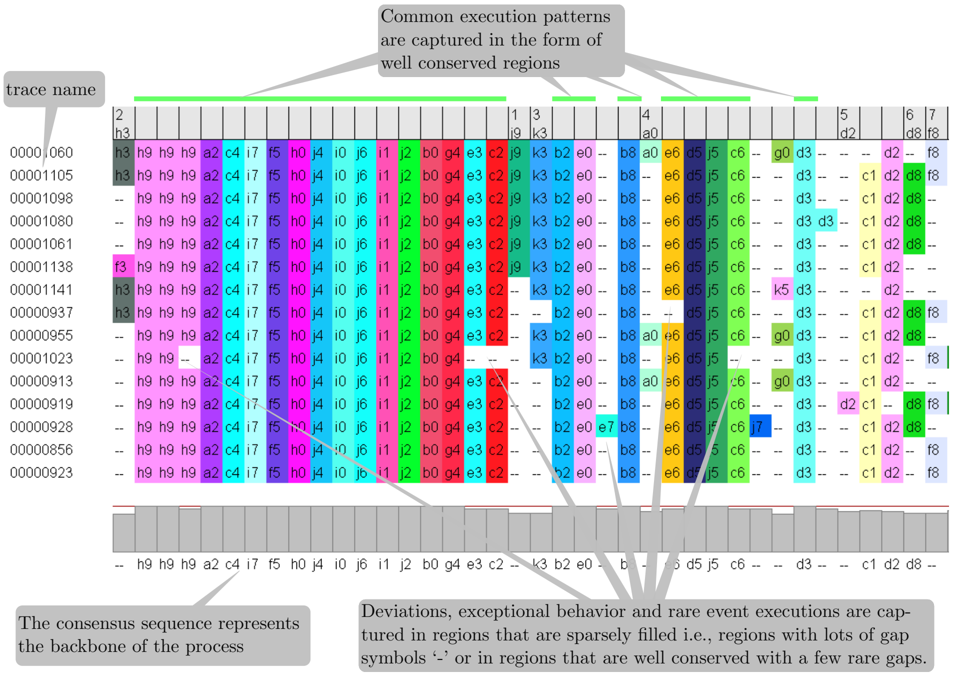

Common execution patterns are captured in the form of well conserved regions
trace name
The consensus sequence represents the backbone of the process
Deviations, exceptional behavior and rare event executions are captured in regions that are sparsely filled i.e., regions with lots of gap symbols '-' or in regions that are well conserved with a few rare gaps.

Figure 11.18: Initial portion of the aligned traces pertaining to the non-urgent cases diagnosed with M13 and whose treatment code is 803. Each row refers to a process instance (a patient case). Columns describe positions in traces. Consider now the cell in row y and column x. If the cell contains an activity name `a`, then `a` occurred for case y at position x. If the cell contains no activity name (i.e., a gap "-"), then nothing happened for y at position x. Trace alignment aims a minimizing the number of gaps and maximizing the consensus.

- one of the cases (trace 00000937) does not have the activity `e6` (squamous cell carcinoma using eia); similarly, one of the cases (trace 00001023) does not execute the activity `c6` (red cell antibody screening)

Such rare behavior/exceptions/deviations can be acceptable or can indicate a violation of expected behavior. For example, skipping a test corresponding to the estimation of the ABO blood group and Rh factor or red cell antibody screening is more likely to be a violation than an acceptable behavior. On the contrary, the rare execution of the cephalin time-coagulation test can be acceptable; based on the history of the patient, this test could have been recommended.

We observed an issue with respect to the recording of events in the event log. The timestamps of events are recorded at the granularity of a day and not at the level of hours/minutes. The side effect of this is that the events in the log could have been reordered, i.e., the order of events may not be reliable. However, this is confined to events within a single day. All events that happen on a day will definitely be logged on the same day. For example, the activity `d2` corresponding to (First outpatient consultation) occurs quite late in the traces (third column from the right in Figure 11.18). However, one would expect it to be one of the initial activities before any tests are performed on the patients. Nonetheless, we can take heart with the consistency in logging as can be seen with the consensus in the alignment (though the ordering of activities is not reliable, we see certain regularity– the activity `d2` occurs in all the traces almost at the same position).

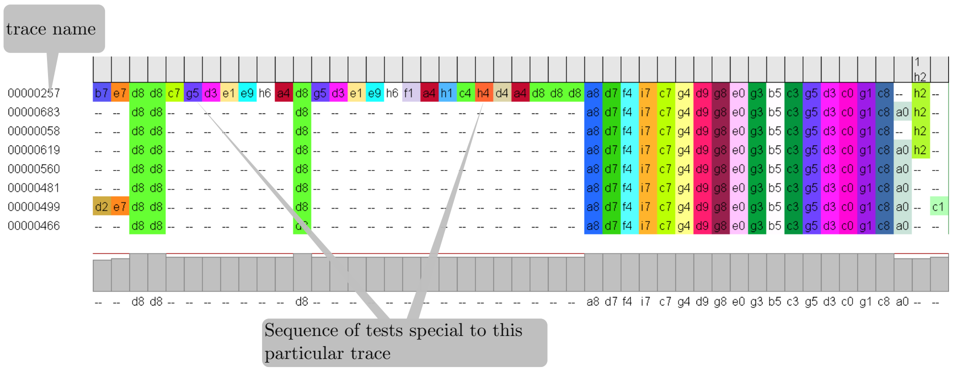


Figure 11.19: Initial portion of the aligned traces pertaining to the urgent cases diagnosed with M13 and whose treatment code is 803.

Figure 11.19 depicts the initial portion of the aligned traces pertaining to the cases in the urgent event log (again we focus on cases diagnosed with M13 and treatment code 803). Common patterns of execution and deviations can be clearly seen. For example, from the alignment, we can see that

- unlike the other urgent cases, the first trace (00000257) has a special execution of a sequence of activities at the beginning. Since the patients are referred to by other hospitals, preliminary sets of diagnosis tests would have already been performed. The doctors at AMC make use of those tests and results. However, in some cases, they might go for a fresh set of tests. This trace corresponds to one such case.
- only one of the traces (trace 00000499) has the activity `c1` corresponding to digoxin.
- two cases (traces 00000257 and 00000058) do not contain the activity `a0` corresponding to CEA - tumor marker using meia. It is expected that these tests are performed. This missing of this test in these traces indicates a potential violation.

In this fashion, trace alignment can assist in uncovering extremely interesting insights and act as probes when analyzing process execution behavior thereby giving cues on process improvement opportunities.

To summarize, though the event log seems to be complex, in reality, it is not. Adopting the systematic approach presented in this section, we realized/showed that the processes are in fact very simple and sequential. Furthermore, based on trace alignment, we noticed that not only are the processes simple and sequential but also the cases share a lot in common with very little deviations from the main path. This case study substantiates our claim that systematic preprocessing of event logs leading to log simplification is crucial in gaining meaningful insights from large scale logs. Unfortunately, preprocessing is less studied/reported in process mining literature.

11.2 Concept Drift in Processes in a Dutch Municipality

Our second case study pertains to detecting concept drifts in processes. We consider three processes from a large municipality in the Netherlands. As discussed in Chapter 5, operational processes need to change to adapt to changing circumstances, e.g., new legislation, extreme variations in supply and demand, etc. For example, since October 1, 2010, the All-in-one Permit for Physical Aspects (omgevingsvergunning) has come into force through the WABO act [247]. This entails an overarching procedure for granting permission for projects like the construction, alteration or use of a house or building, etc. Now, the municipalities have one permit, one procedure and one set of submittal requirements, followed by one legal remedies procedure and enforcement by one authority.

Municipalities are interested in getting insights into their processes, e.g., the way they are planned to be executed vis-a-vis the way they are actually executed. Moreover, they want to know which parts/regions in processes are time consuming. Municipalities find such insights important and interesting for many reasons. For example, in some cases, municipalities can only charge its customers based on the real costs for providing a service and in other cases they can charge a fixed fee for a service. Also, in case of permit requests, if the municipalities do not come to a decision within a certain time (as set by the law) then the permit has to be granted. Therefore, they want to be as cost efficient as possible. Furthermore, processes within different municipalities are very similar in many aspects. At the same time, each municipality can have its own characteristics (e.g., differences in size, demographics, problems, and policies) that need to be maintained. Recently, different municipalities in the Netherlands have evinced interest in comparing their processes and learning from each other (the interested reader is referred to the CoSeLog project [45] for further information.). Their vision is to have a form of standardization through a centrally managed process management system [28, 219, 220, 250]. This includes the definition of configurable process models allowing for variations peculiar to each municipality. The configurable process model can be realized from a set of concrete models (normative models) that capture the desired or required behavior. However, more often than not, the concrete models are either not available or even if available are of very low quality. Process mining plays a significant role in bridging this gap by enabling the discovery of what the actual processes are. One can consider both normative models as well as discovered models in extracting configurable process models. This makes the discovery of good and correct process models (reflecting the reality) from event logs extremely crucial. When analyzing event logs, one needs to factor in the possibility of process changes, i.e., *concept drifts*, that could have taken place. In the following sections, we present the results of analysis of concept drifts in event logs pertaining to three processes from this municipality related to building permits.

11.2.1 Permit Process for Temporary Rental of Vacant Dwellings

Our first process corresponds to obtaining permits for temporary rental of vacant dwellings. If a person wants to rent out unoccupied dwellings, a permit from the municipal authorities is required. A permit is sanctioned, usually valid for a period of two years, if they satisfy a number of conditions.

We considered an event log containing 35 cases and 315 events referring to 10 activities. The cases pertain to permit requests for temporary rental of vacant dwellings spanning over the period between 16-04-2009 and 05-01-2011. We considered the window count feature on the *follows* relation for all activity pairs using a window of size 4. Since the mean trace length is small (9), we considered a smaller window size for feature extraction. The window count feature of each activity pair defines a vector of size 35, corresponding to the traces in the event log. The univariate Kolmogorov-Smirnov test (KS-test) is applied on each of these vectors using a population size of 6 (since we have only 35 traces, we considered smaller populations). Figure 11.20 depicts the average significance probability of the KS-test on all activity pairs. We see two troughs formed at indices 11 and 17. These troughs signify a change in behavior in the traces preceding and succeeding them. Figure 11.20 also depicts the start timestamps (24-11-2009 and 12-02-2010 respectively) of the cases corresponding to these troughs.

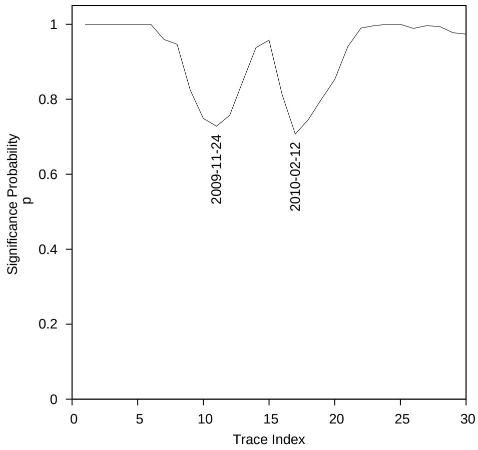


Figure 11.20: Average significance probability (over all activity pairs) of KS-test on window count measure. The population size for the KS-test is 6. There are two troughs signifying a change in behavior.

Considering the two change points, we split the log into three partitions, the first, $\mathcal{L}_1$,

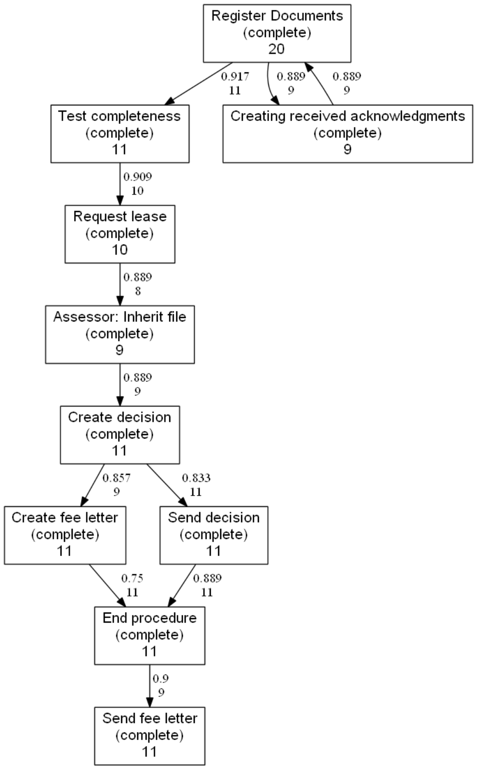
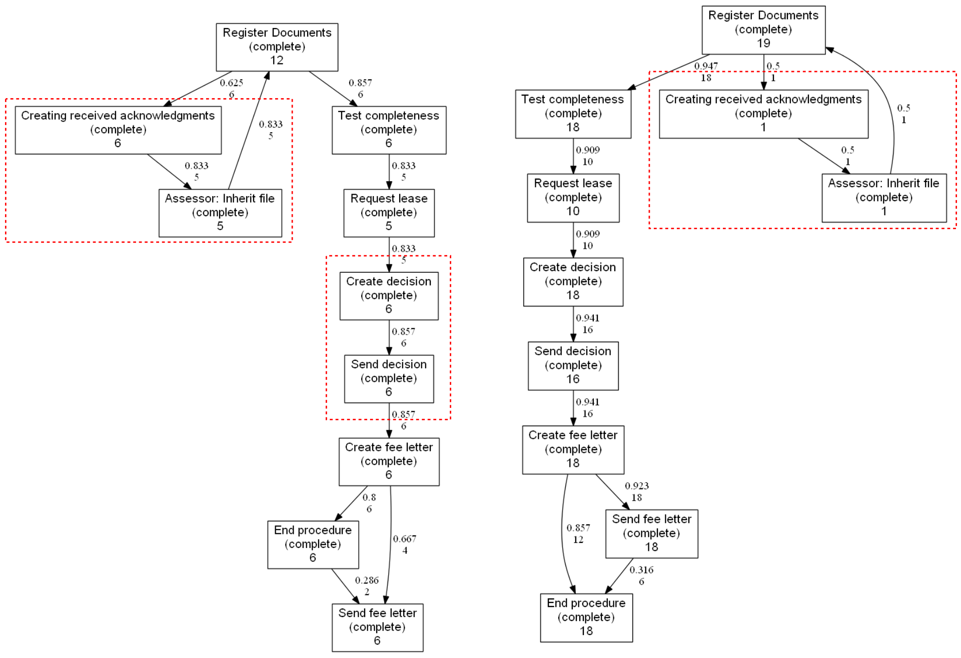

Register Documents (complete) 20
0.917 11
0.889 9
0.889 9
Test completeness (complete) 11
Creating received acknowledgments (complete) 9
0.909 10
Request lease (complete) 10
0.889 8
Assessor: Inherit file (complete) 9
0.889 9
Create decision (complete) 11
0.857 9
0.833 11
Create fee letter (complete) 11
Send decision (complete) 11
0.75 11
0.889 11
End procedure (complete) 11
0.9 9
Send fee letter (complete) 11

Figure 11.21: Heuristic net of the permit process for temporary rental of vacant dwellings discovered using the event log $\mathcal{L}_1$.

containing the traces from the beginning until the first change point (i.e., traces 1 to 11), the second, $\mathcal{L}_2$, containing the traces between the first and second change points (i.e., traces 12 to 17), and the third, $\mathcal{L}_3$, containing the traces from the second change point until the end (i.e., traces 18 to 35). Figure 11.21 depicts the process model discovered using the Heuristics miner [264] on the event log $\mathcal{L}_1$. The documents submitted by the applicant are first registered at the municipality (by an employee). The municipality notifies the applicant of the receipt of the documents and tests for its completeness. The municipality requests for the lease (rental agreement) of the vacant dwelling (if needed) and a decision is taken and communicated to the applicant. A fee letter is also prepared and sent. The preparation of the fee letter can happen before/after the decision is sent; this is captured in the parallel construct in Figure 11.21. The sending of the fee letter can happen before/after the End procedure activity.

Figure 11.22(a) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_2$. There are two changes (marked with dashed rectangles) in this model when compared to the model in Figure 11.21. Firstly, the activity Assessor: Inherit file related to Creating received acknowledgments has to be executed before

Test completeness. Secondly, the creation of fee letter can happen only after the decision is sent. The process owners indeed acknowledged that their permit process has changed in November 2009, thus, validating our change point detection. Figure 11.22(b) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_3$. The figure also depicts the region corresponding to the change when compared to Figure 11.22(a). Unlike the process in Figure 11.22(a), the municipality now does not send acknowledgements for each and every permit request. Out of the 18 permit requests only one of them was sent an acknowledgment (an applicant is allowed an option to indicate that a confirmation is not needed). Such measures are typically taken to reduce costs.


Register Documents (complete) 12
0.625 6
0.857 6
Creating received acknowledgments (complete) 6
0.833 5
Test completeness (complete) 6
0.833 5
Assessor: Inherit file (complete) 5
0.833 5
Request lease (complete) 5
0.833 5
Create decision (complete) 6
0.857 6
Send decision (complete) 6
0.857 6
Create fee letter (complete) 6
0.8 6
End procedure (complete) 6
0.667 4
0.286 2
Send fee letter (complete) 6
Register Documents (complete) 19
0.947 18
0.5 1
Test completeness (complete) 18
Creating received acknowledgments (complete) 1
0.5 1
0.909 10
Request lease (complete) 10
0.5 1
Assessor: Inherit file (complete) 1
0.909 10
Create decision (complete) 18
0.941 16
Send decision (complete) 16
0.941 16
Create fee letter (complete) 18
0.923 18
0.857 12
Send fee letter (complete) 18
0.316 6
End procedure (complete) 18

(a) event log $\mathcal{L}_2$ (b) event log $\mathcal{L}_3$

Figure 11.22: Heuristic nets of the permit process for temporary rental of vacant dwellings discovered using the event logs $\mathcal{L}_2$ and $\mathcal{L}_3$. The dashed rectangles in (a) highlight the regions corresponding to the change in the process with respect to the process model in Figure 11.21. The dashed rectangle in (b) highlights the region corresponding to the change in the process with respect to the model depicted in (a).

Figure 11.23(a) depicts the average significance probability of the KS-test over all activity pairs for the same event log using the J-measure as the feature computed on a window of size 4. Unlike Figure 11.20, we see only one trough. This can be attributed to the reliance of J-measure on the probability of activities. Since the activities Creating received acknowledgments and Assessor: Inherit file are rarely executed in the traces 18 to 35, the probability of activities is significantly different from that of the activities in the traces 1 to 17. This is captured in the trough at index 17. The probability of activities dominates the probability of the follows relation between activity pairs and hence we do not see a prominent change at index 11. Furthermore, since the significance probabilities are averaged over all activity pairs, the change at index 11 being confined to a few activities is obscured by the others. Nonetheless, we can see a minor dip at index 11 (as indicated by the arrow) in Figure 11.23(a). Figure 11.23(b) depicts the significance probability of the KS-test on the J-measure for the *follows* relation for the activity pair (Register Documents, Create Decision). We can now see two drifts signifying change points at indices 11 and 17. The change point at index 11 indicates that there is a change in the process in the region between Register Documents and Create Decision, which is indeed the case.

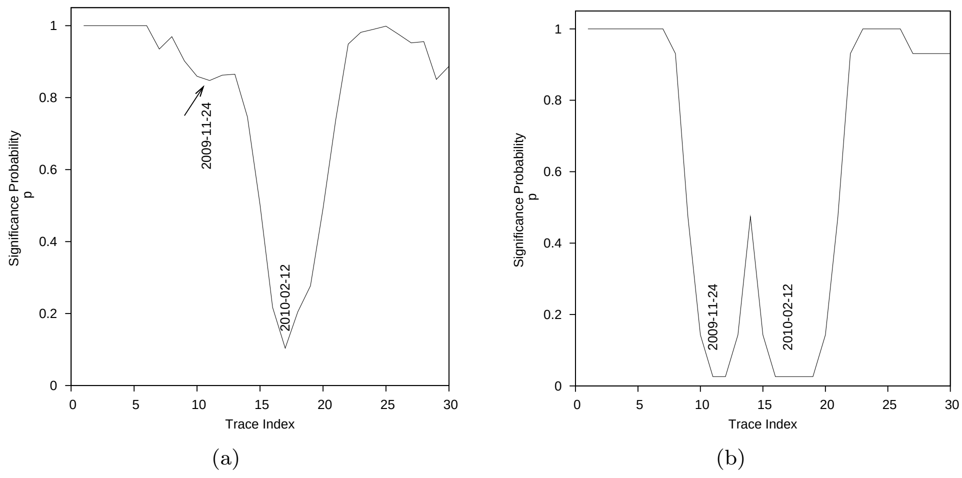


Figure 11.23: (a) Average significance probability (over all activity pairs) of KS-test on the J-measure. The population size for the KS-test is 6. There is one trough signifying a change in behavior. (b) Significance probability of the KS-test on the J-measure for the *follows* relation for the activity pair (Register Documents, Create Decision).

11.2.2 Permit Process for Advertisements

We analyzed various other processes within the same municipality. We also investigated advertisement permits. If a person/organization wants to advertise on a building in the Netherlands, for example on a billboard or an illuminated sign, a permit is needed in most cases, which can be obtained from the local municipality. The municipality may charge an advertising tax or municipal tax on encroachments on or above public space (precariorechten) for advertisements visible from the public road.

We considered an event log containing 116 cases and 2335 events distributed over 25 activities. The cases pertain to permit requests for placing advertisements spanning over the period between 07-07-2003 and 18-03-2008. We considered the J-measure feature on the *follows* relation for all activity pairs using a window of size 10. The J-measure values of each activity pair define a vector of size 116, corresponding to the traces in the event log. The univariate Kolmogorov-Smirnov test (KS-test) is applied on each of these vectors using a population size of 10. Figure 11.24 depicts the average significance probability of the KS-test on all activity pairs. We see four troughs formed at indices 42, 74, 84, and 103. These troughs signify a change in behavior in the traces preceding and succeeding them. Among the four troughs, the one at index 42 is particularly significant. Figure 11.24 also depicts the start timestamps (04-10-2004, 27-10-2005, 13-02-2006, and 31-08-2006 respectively) of the cases corresponding to these troughs.

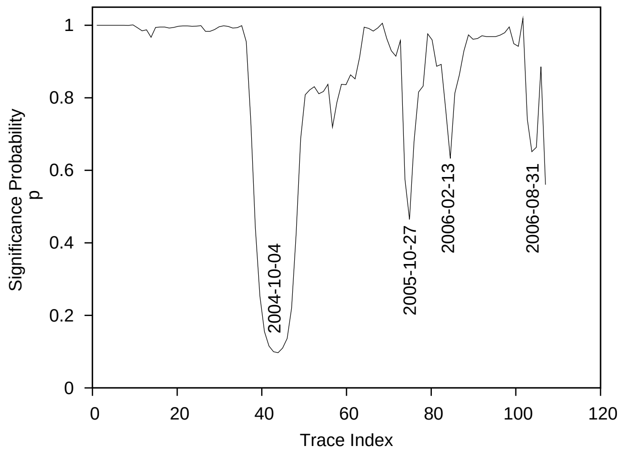


Figure 11.24: Average significance probability (over all activity pairs) of KS-test on J-measure. The population size for the KS-test is 10. There are four troughs signifying a change in behavior.

Based on the four change points, we split the log into five partitions, the first, $\mathcal{L}_1$, containing the traces from the beginning until the first change point (i.e., traces 1 to 42), the second, $\mathcal{L}_2$, containing the traces between the first and second change points (i.e., traces 43 to 74), the third, $\mathcal{L}_3$, containing the traces from the second change point until the third change point (i.e., traces 75 to 84), the fourth, $\mathcal{L}_4$, containing the traces from the third change point to the fourth change point (i.e., traces 85 to 103), and the fifth, $\mathcal{L}_5$, containing the traces from the fourth change point until the end of the log (i.e., traces 104 to 116). Figure 11.25 depicts the process model discovered using the Heuristics miner [264] on the event log $\mathcal{L}_1$. The process can be divided into four high-level sub-procedures as depicted in the figure and listed below.

- Upon submission of an application, the municipality acknowledges the receipt of documents and (optionally) tests for its completeness.
- The municipality then proceeds with a follow-up procedure that verifies whether the application and submitted documents are in compliance with the regulations.

- Based on the investigations, the municipality then makes a decision on the application and informs the applicant with the decision along with a fee letter.
- Finally, the municipality registers the advertisements placed and enforces them.

Figure 11.26 depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_2$. The figure highlights regions that differ from the process model in Figure 11.25. There are two changes in this model with respect to the previous one. The first change is related to the checking for completeness of the registered documents. In the initial process model (Figure 11.25), this check was not mandatory (only 2 of the 43 applications were checked for completeness). The municipality changed this process by making the checks mandatory before proceeding. The second change is the introduction of a new activity End procedure: enforcement is next as highlighted in Figure 11.26. The initial process model had only the activity End procedure, possibly choose enforcement where as the new model has both these activities.

Figure 11.27(a) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_3$. This model contains only one type of enforcement activity End procedure: enforcement is next indicating that the municipality has phased out the activity End procedure, possibly choose enforcement. Figure 11.27(b) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_4$. The change corresponds to the region marked in the figure involving the activities Administration: copy file and Assign to supervisor. Unlike the previous model where these activities happen in a sequence, they can now be executed concurrently. Figure 11.28 depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_5$. In all of the previous models, the activity Control advertisements placed is executed after the initiation of the enforcement procedure. However, in the model based on $\mathcal{L}_5$, this activity can execute concurrently with the administrative activities once a decision has been taken.

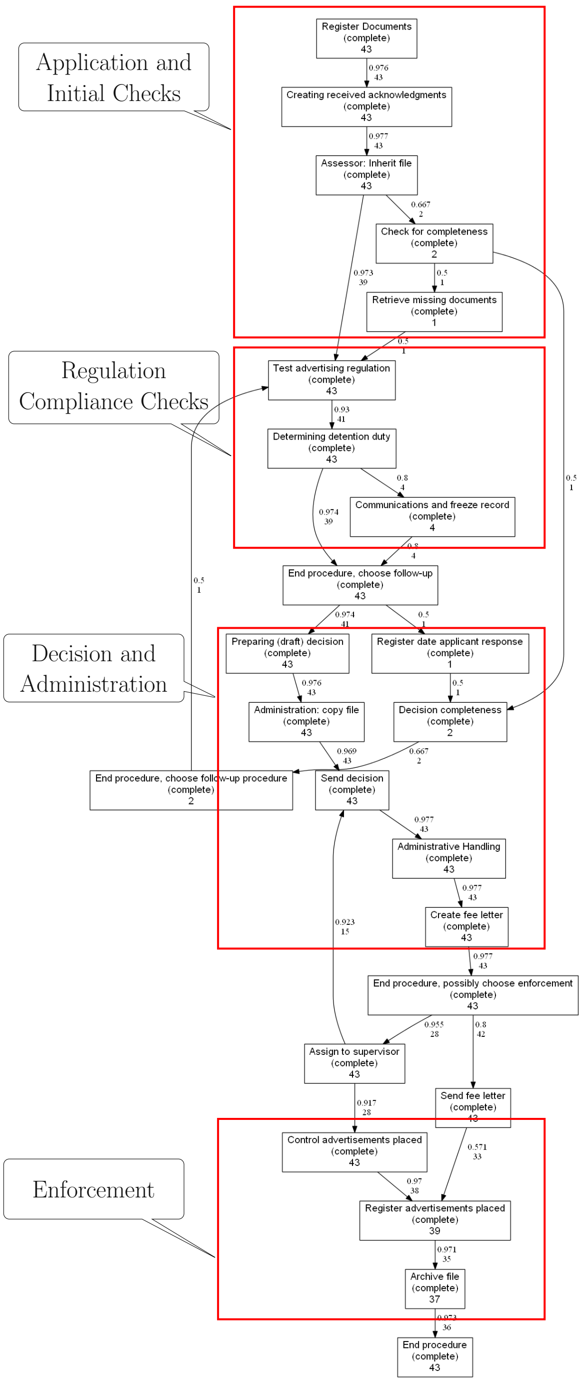

Application and Initial Checks
Regulation Compliance Checks
Decision and Administration
Enforcement
Register Documents (complete) 43
0.976 43
Creating received acknowledgments (complete) 43
0.977 43
Assessor: Inherit file (complete) 43
0.667 2
Check for completeness (complete) 2
0.973 39
0.5 1
Retrieve missing documents (complete) 1
0.5 1
Test advertising regulation (complete) 43
0.93 41
Determining detention duty (complete) 43
0.8 4
Communications and freeze record (complete) 4
0.974 39
0.8 4
0.5 1
End procedure, choose follow-up (complete) 43
0.974 41
0.5 1
Preparing (draft) decision (complete) 43
Register date applicant response (complete) 1
0.976 43
0.5 1
Administration: copy file (complete) 43
Decision completeness (complete) 2
0.969 43
0.667 2
End procedure, choose follow-up procedure (complete) 2
Send decision (complete) 43
0.977 43
Administrative Handling (complete) 43
0.977 43
Create fee letter (complete) 43
0.923 15
0.977 43
End procedure, possibly choose enforcement (complete) 43
0.955 28
0.8 42
Assign to supervisor (complete) 43
Send fee letter (complete) 43
0.917 28
Control advertisements placed (complete) 43
0.571 33
0.97 38
Register advertisements placed (complete) 39
0.971 35
Archive file (complete) 37
0.973 36
End procedure (complete) 43

Figure 11.25: Heuristic net of the permit process for advertisements discovered using the event log $\mathcal{L}_1$. The marked regions depict high-level sub-procedures in this process.

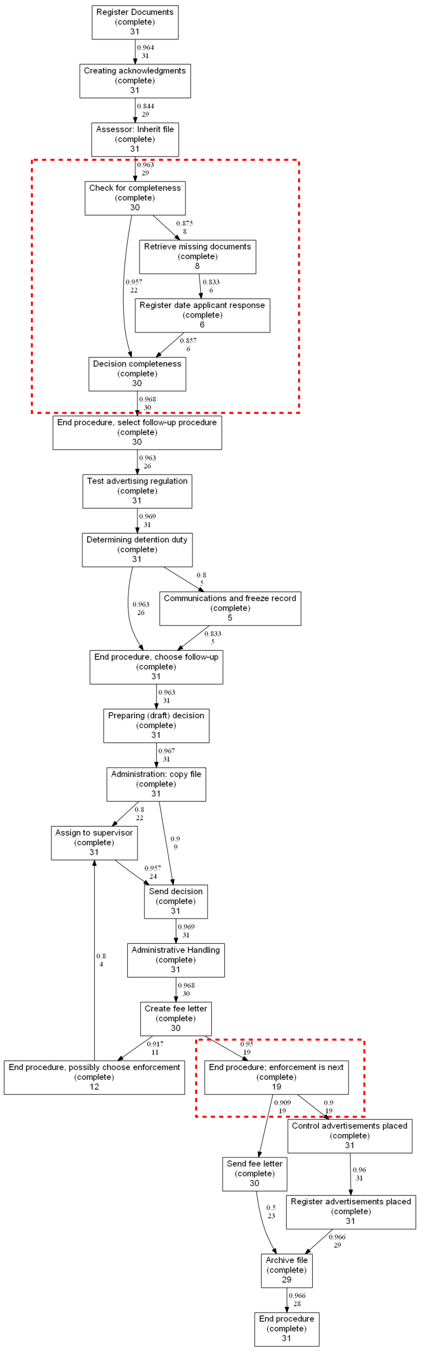
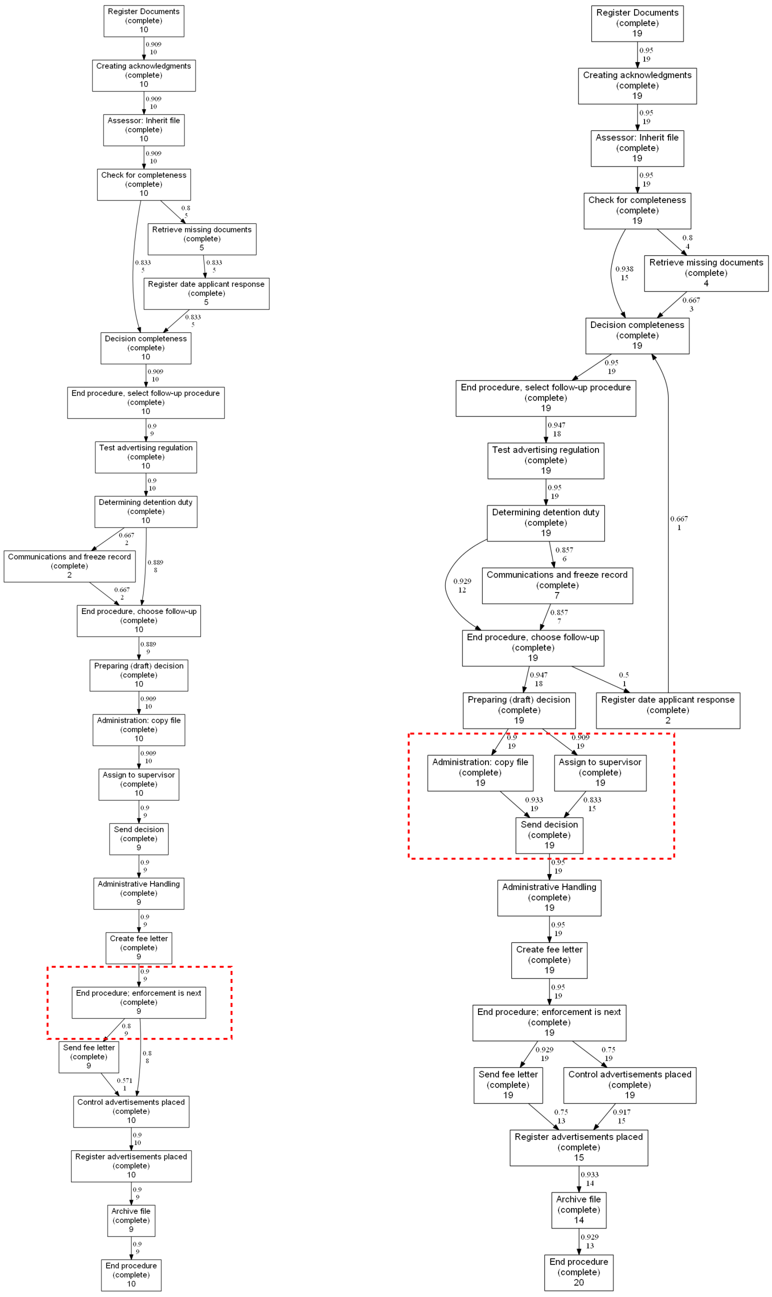

Register Documents (complete) 31
Creating acknowledgments (complete) 31
Assessor: Inherit file (complete) 31
Check for completeness (complete) 30
Retrieve missing documents (complete) 8
Register date applicant response (complete) 6
Decision completeness (complete) 30
End procedure, select follow-up procedure (complete) 30
Test advertising regulation (complete) 31
Determining detention duty (complete) 31
Communications and freeze record (complete) 5
End procedure, choose follow-up (complete) 31
Preparing (draft) decision (complete) 31
Administration: copy file (complete) 31
Assign to supervisor (complete) 31
Send decision (complete) 31
Administrative Handling (complete) 31
Create fee letter (complete) 30
End procedure, possibly choose enforcement (complete) 12
End procedure; enforcement is next (complete) 19
Control advertisements placed (complete) 31
Send fee letter (complete) 30
Register advertisements placed (complete) 31
Archive file (complete) 29
End procedure (complete) 31

Figure 11.26: Heuristic net of the permit process for advertisements discovered using the event log $\mathcal{L}_2$. The marked regions depict regions corresponding to the change in the process when compared to the model in Figure 11.25. The municipality has now made the checks for completeness mandatory and introduced a new activity End procedure, enforcement is next.


Register Documents (complete) 10
Creating acknowledgments (complete) 10
Assessor: Inherit file (complete) 10
Check for completeness (complete) 10
Retrieve missing documents (complete) 5
Register date applicant response (complete) 5
Decision completeness (complete) 10
End procedure, select follow-up procedure (complete) 10
Test advertising regulation (complete) 10
Determining detention duty (complete) 10
Communications and freeze record (complete) 2
End procedure, choose follow-up (complete) 10
Preparing (draft) decision (complete) 10
Administration: copy file (complete) 10
Assign to supervisor (complete) 10
Send decision (complete) 9
Administrative Handling (complete) 9
Create fee letter (complete) 9
End procedure; enforcement is next (complete) 9
Send fee letter (complete) 9
Control advertisements placed (complete) 10
Register advertisements placed (complete) 10
Archive file (complete) 9
End procedure (complete) 10
Register Documents (complete) 19
Creating acknowledgments (complete) 19
Assessor: Inherit file (complete) 19
Check for completeness (complete) 19
Retrieve missing documents (complete) 4
Decision completeness (complete) 19
End procedure, select follow-up procedure (complete) 19
Test advertising regulation (complete) 19
Determining detention duty (complete) 19
Communications and freeze record (complete) 7
End procedure, choose follow-up (complete) 19
Preparing (draft) decision (complete) 19
Register date applicant response (complete) 2
Administration: copy file (complete) 19
Assign to supervisor (complete) 19
Send decision (complete) 19
Administrative Handling (complete) 19
Create fee letter (complete) 19
End procedure; enforcement is next (complete) 19
Send fee letter (complete) 19
Control advertisements placed (complete) 19
Register advertisements placed (complete) 15
Archive file (complete) 14
End procedure (complete) 20

(a) event log $\mathcal{L}_3$ (b) event log $\mathcal{L}_4$

Figure 11.27: Heuristic nets of the permit process for advertisements discovered using the event logs $\mathcal{L}_3$ and $\mathcal{L}_4$. The activity End procedure, possibly choose enforcement has been phased out in (a) when compared to the model in Figure 11.26. Unlike the model in (a), the activities Administration: copy file and Assign to supervisor can be executed in parallel in (b).

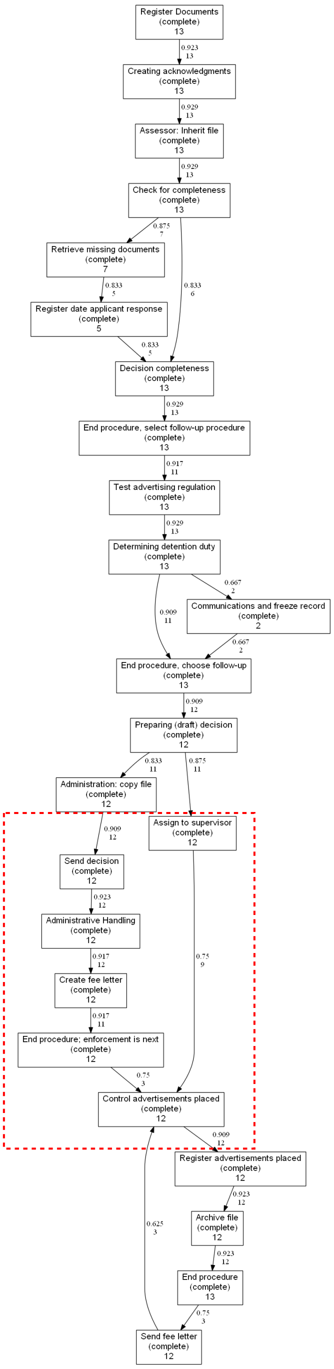

Register Documents (complete) 13
0.923 13
Creating acknowledgments (complete) 13
0.929 13
Assessor: Inherit file (complete) 13
0.929 13
Check for completeness (complete) 13
0.875 7
Retrieve missing documents (complete) 7
0.833 5
0.833 6
Register date applicant response (complete) 5
0.833 5
Decision completeness (complete) 13
0.929 13
End procedure, select follow-up procedure (complete) 13
0.917 11
Test advertising regulation (complete) 13
0.929 13
Determining detention duty (complete) 13
0.667 2
Communications and freeze record (complete) 2
0.909 11
0.667 2
End procedure, choose follow-up (complete) 13
0.909 12
Preparing (draft) decision (complete) 12
0.833 11
0.875 11
Administration: copy file (complete) 12
Assign to supervisor (complete) 12
0.909 12
Send decision (complete) 12
0.923 12
Administrative Handling (complete) 12
0.75 9
0.917 12
Create fee letter (complete) 12
0.917 11
End procedure; enforcement is next (complete) 12
0.75 3
Control advertisements placed (complete) 12
0.909 12
Register advertisements placed (complete) 12
0.923 12
0.625 3
Archive file (complete) 12
0.923 12
End procedure (complete) 13
0.75 3
Send fee letter (complete) 12

Figure 11.28: Heuristic net of the permit process for advertisements discovered using the event log $\mathcal{L}_5$. Unlike the previous model in Figure 11.27(b), the controlling of advertisements placed can now happen in parallel with the administrative activities once a decision has been taken.

11.2.3 Permit Process for Driveway Construction

To further illustrate our approach to discover concept drift, we consider the process of obtaining a permit to build or change a driveway from ones premises to the municipality's public road, referred to as a 'driveway permit' (inritvergunning) or as an 'egress permit' (uitwegvergunning). If a person (business) wants to build a driveway road to a provincial highway, permission from the provincial authority is needed. The municipality or provincial authority considers issues such as safe and efficient road use, protection of green spaces, and protection of the ambience of the surrounding areas before sanctioning a permit. The permit application must be accompanied with the necessary fees and documentation, including design plans, photos, and pertinent reports.

We considered an event log containing 315 cases and 3968 events referring to 21 activities. The cases pertain to permit requests for driveway construction spanning over the period between 03-01-2006 and 12-01-2011. We considered the J-measure feature on the *follows* relation for all activity pairs using a window of size 10. The J-measure values of each activity pair define a vector of size 315, corresponding to the traces in the event log. The univariate Kolmogorov-Smirnov test (KS-test) is applied on each of these vectors using a population size of 20. Figure 11.29 depicts the average significance probability of the KS-test on all activity pairs. We see seven troughs formed at indices 21, 91, 125, 168, 213, 237, and 291 respectively. These troughs signify a change in behavior in the traces preceding and succeeding them. Based on these change points, we split the log into 8 partitions as depicted in the figure.

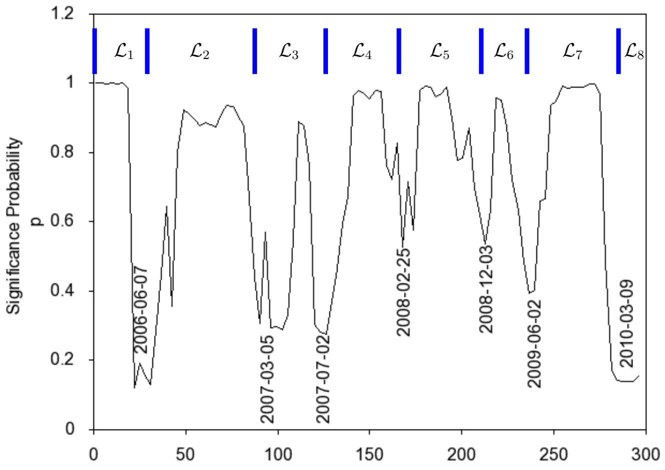


Figure 11.29: Average significance probability (over all activity pairs) of KS-test on J-measure. The population size for the KS-test is 20. There are seven troughs signifying a change in behavior. These are used to partition the log into $\mathcal{L}_1$, $\mathcal{L}_2$, ..., $\mathcal{L}_8$.

Figure 11.30 depicts the process model discovered using the Heuristics miner [264]

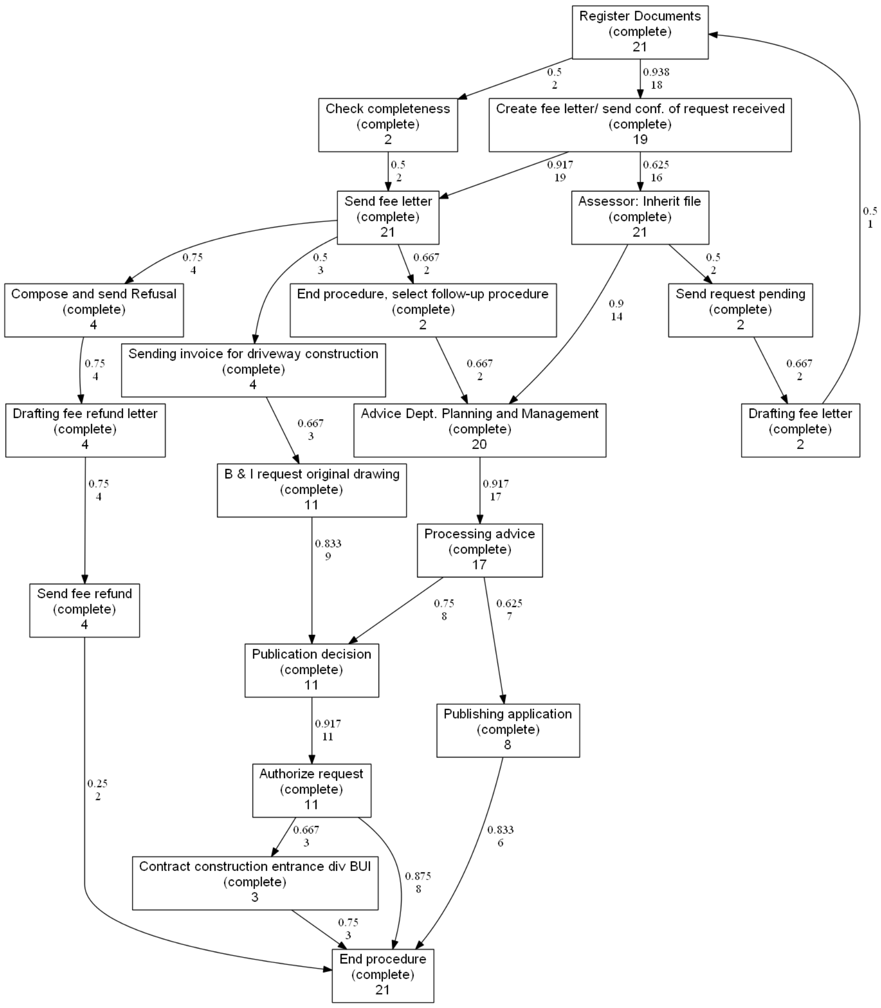

Register Documents (complete) 21
Check completeness (complete) 2
Create fee letter/ send conf. of request received (complete) 19
Send fee letter (complete) 21
Assessor: Inherit file (complete) 21
Compose and send Refusal (complete) 4
End procedure, select follow-up procedure (complete) 2
Send request pending (complete) 2
Sending invoice for driveway construction (complete) 4
Drafting fee refund letter (complete) 4
Advice Dept. Planning and Management (complete) 20
Drafting fee letter (complete) 2
B & I request original drawing (complete) 11
Processing advice (complete) 17
Send fee refund (complete) 4
Publication decision (complete) 11
Publishing application (complete) 8
Authorize request (complete) 11
Contract construction entrance div BUI (complete) 3
End procedure (complete) 21

Figure 11.30: Heuristic net of the permit process for driveway construction discovered using the event log $\mathcal{L}_1$.

on the event log $\mathcal{L}_1$. Upon submission of an application, the municipality acknowledges the receipt of documents and (optionally) tests for its completeness. A fee letter is also prepared and sent to the applicant. The municipality then proceeds with a follow-up procedure where an opinion from the department of planning and management is sought. If needed, the municipality may ask for the original drawings of the planned driveway. Based on the investigations, the municipality then makes a decision on the application and publishes them. The municipality may refuse the permission in which case a fee refund letter is sent to the applicant. For some requests that have been authorized, the municipality may send an invoice for driveway construction and contract the BUI division for construction. The BUI division may in turn confirm its decision to the municipality.

Figure 11.31 depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_2$. There are two primary changes (marked by the dashed rectangles) in this process model when compared to the model in Figure 11.30. Firstly, the checking for completeness of the registered documents is no longer optional. Also, unlike the previous model where there are two activities related to creating of a fee letter, viz., Create fee letter/send conf. of request received and Drafting fee letter, we now have only one activity Drafting fee letter. In addition, the confirmation of requests is no longer sent to the applicants. The second change corresponds to the omission of the activity Publishing application. Figure 11.32 depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_3$. There is one behavioral change in this model when compared to the one in Figure 11.31. The municipality has now enforced a stricter need for confirmation from the BUI division. In Figure 11.31, only 5 of the 24 contracts submitted to the BUI division were confirmed where as in Figure 11.32, 13 of the 14 contracts were confirmed.

Figure 11.33 depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_4$. There are two primary changes (as indicated by the dashed rectangles) in this model when compared to the previous model in Figure 11.32. Firstly, a refund is recorded only to some selected cases in case of refusal of a permit. Secondly, the number of contracts given to the BUI division has drastically reduced. Figure 11.34(a) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_5$. There are no further contracts to the BUI division in this process. Furthermore, the activities Advice Dept. Planning and Management and B & I request original drawing happen more often in parallel with the activities Send fee letter and End procedure, select follow-up procedure when compared to the previous model. Figure 11.34(b) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_6$. In this model, the activities Drafting fee letter and Send request pending can be executed in parallel when compared to the previous models where they happen in a sequence. Furthermore, refunds have been stopped for all applicants in case of permit refusal.

Figure 11.35(a) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_7$. This model differs from the model in Figure 11.34(b) in that the activity Send fee letter has to happen before the End procedure, select follow-up

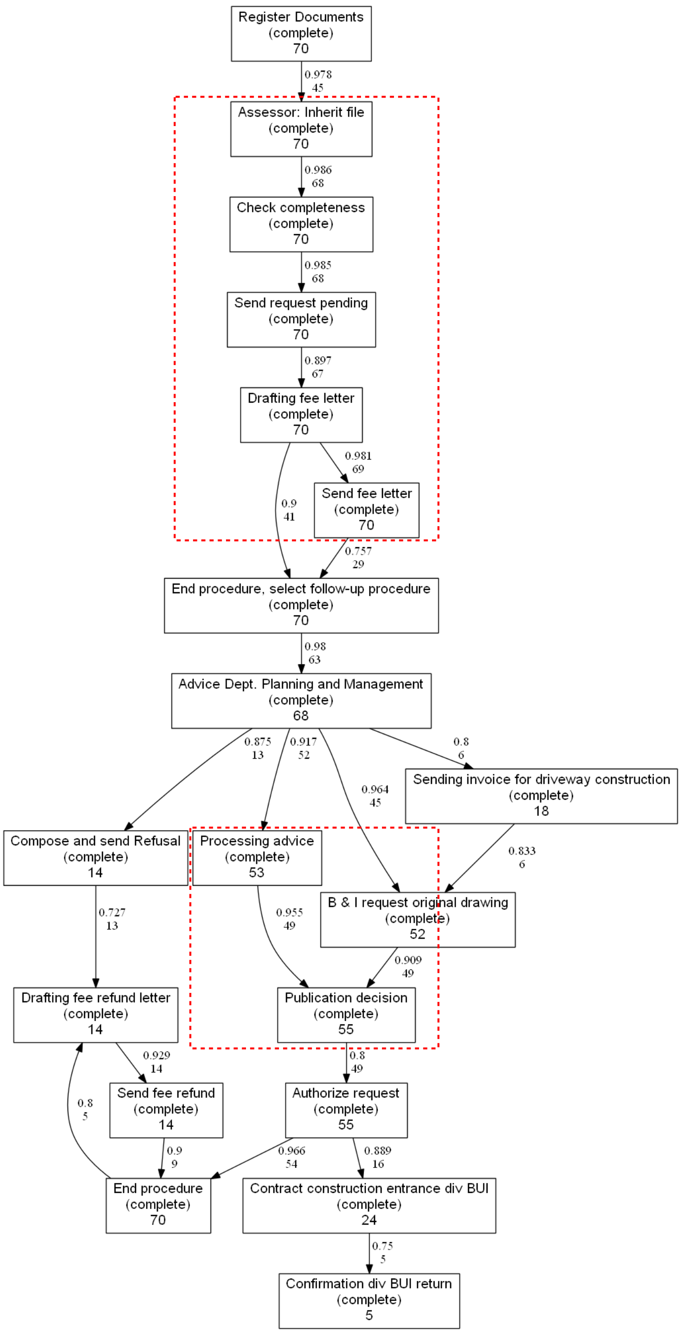

Register Documents (complete) 70
0.978 45
Assessor: Inherit file (complete) 70
0.986 68
Check completeness (complete) 70
0.985 68
Send request pending (complete) 70
0.897 67
Drafting fee letter (complete) 70
0.981 69
0.9 41
Send fee letter (complete) 70
0.757 29
End procedure, select follow-up procedure (complete) 70
0.98 63
Advice Dept. Planning and Management (complete) 68
0.875 13
0.917 52
0.964 45
0.8 6
Sending invoice for driveway construction (complete) 18
Compose and send Refusal (complete) 14
Processing advice (complete) 53
0.833 6
B & I request original drawing (complete) 52
0.727 13
0.955 49
0.909 49
Drafting fee refund letter (complete) 14
Publication decision (complete) 55
0.929 14
0.8 49
0.8 5
Send fee refund (complete) 14
Authorize request (complete) 55
0.9 9
0.966 54
0.889 16
End procedure (complete) 70
Contract construction entrance div BUI (complete) 24
0.75 5
Confirmation div BUI return (complete) 5

Figure 11.31: Heuristic net of the permit process for driveway construction discovered using the event log $\mathcal{L}_2$. The dashed rectangles indicate regions where changes had taken place when compared to the previous model. When compared to the model in Figure 11.30, the tests for completeness is now made mandatory and the confirmation of requests is no longer sent to applicants.

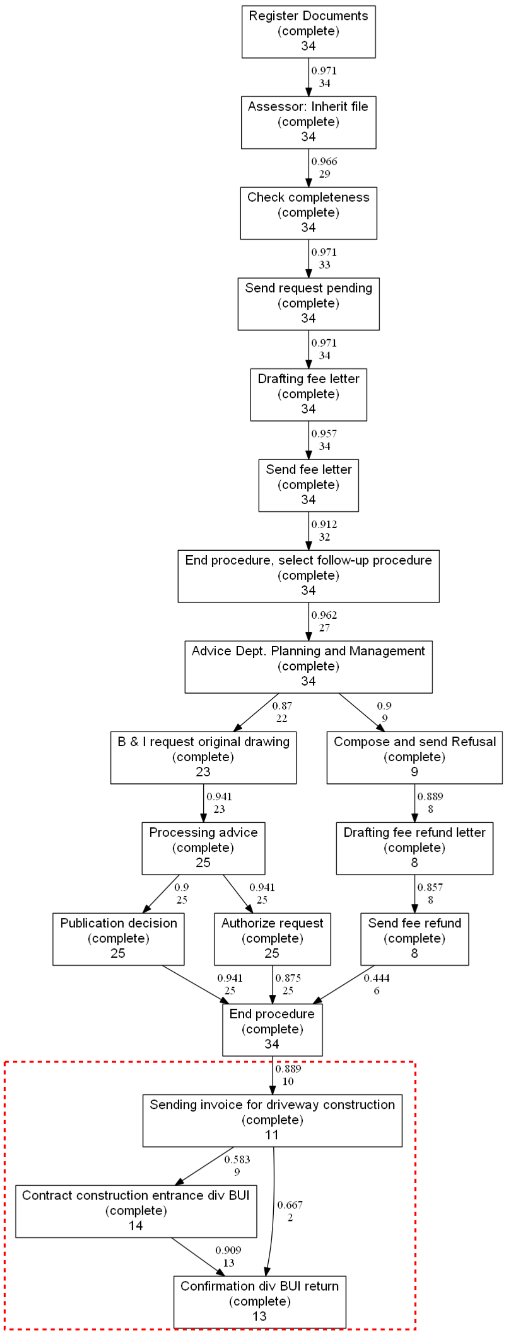

Register Documents (complete) 34
0.971 34
Assessor: Inherit file (complete) 34
0.966 29
Check completeness (complete) 34
0.971 33
Send request pending (complete) 34
0.971 34
Drafting fee letter (complete) 34
0.957 34
Send fee letter (complete) 34
0.912 32
End procedure, select follow-up procedure (complete) 34
0.962 27
Advice Dept. Planning and Management (complete) 34
0.87 22
0.9 9
B & I request original drawing (complete) 23
Compose and send Refusal (complete) 9
0.941 23
0.889 8
Processing advice (complete) 25
Drafting fee refund letter (complete) 8
0.9 25
0.941 25
0.857 8
Publication decision (complete) 25
Authorize request (complete) 25
Send fee refund (complete) 8
0.941 25
0.875 25
0.444 6
End procedure (complete) 34
0.889 10
Sending invoice for driveway construction (complete) 11
0.583 9
0.667 2
Contract construction entrance div BUI (complete) 14
0.909 13
Confirmation div BUI return (complete) 13

Figure 11.32: Heuristic net of the permit process for driveway construction discovered using the event log $\mathcal{L}_3$. There is a stricter need for confirmation from the BUI division in this model when compared to the model in Figure 11.31.

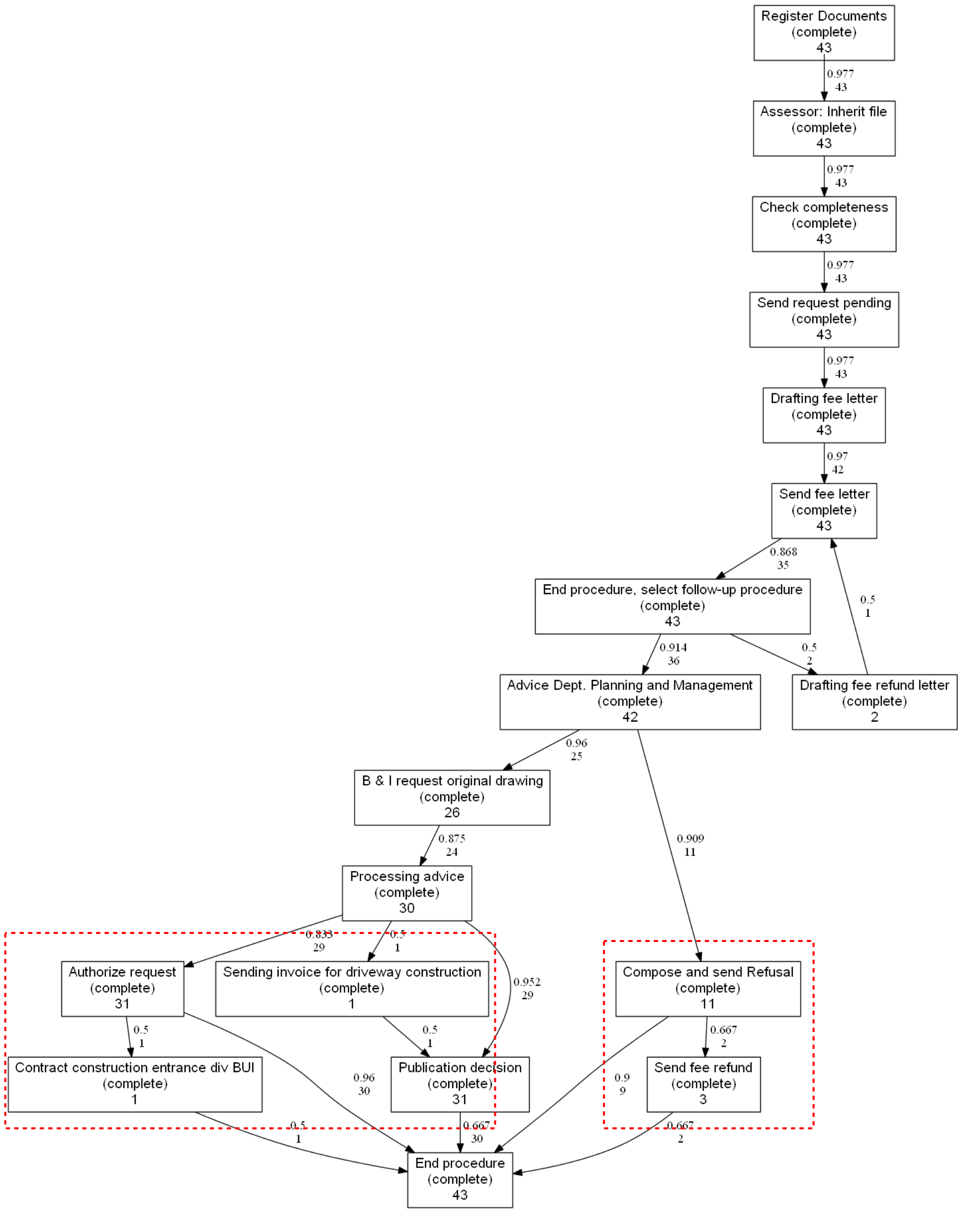
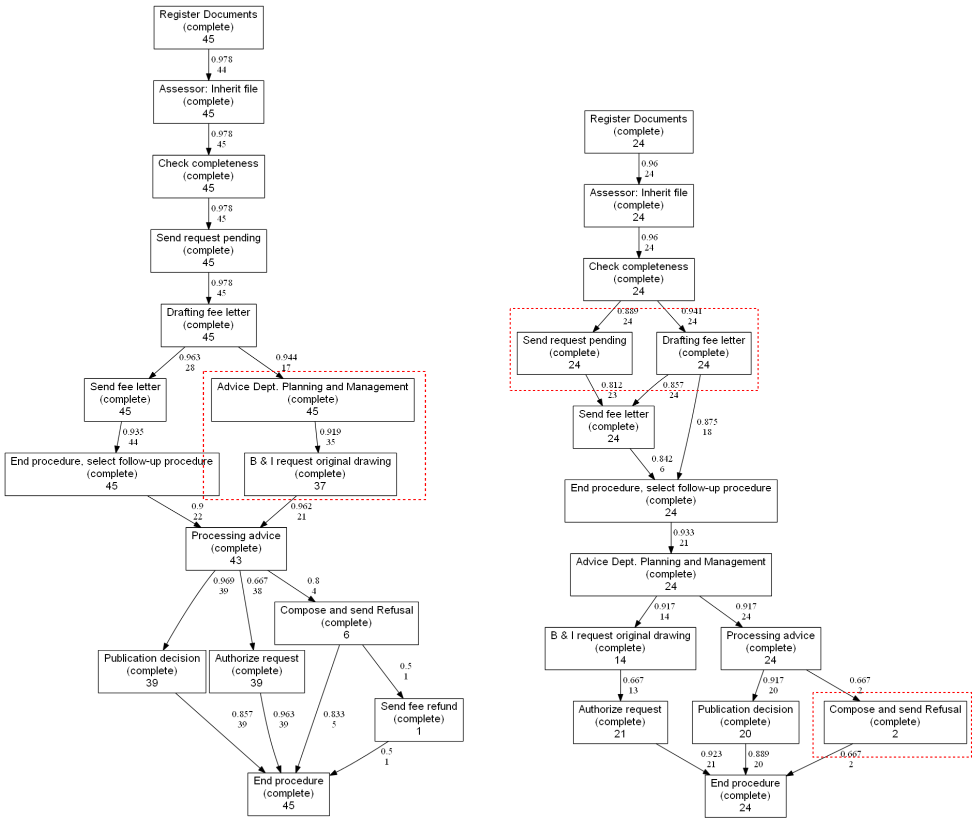

Register Documents (complete) 43
0.977 43
Assessor: Inherit file (complete) 43
0.977 43
Check completeness (complete) 43
0.977 43
Send request pending (complete) 43
0.977 43
Drafting fee letter (complete) 43
0.97 42
Send fee letter (complete) 43
0.868 35
End procedure, select follow-up procedure (complete) 43
0.5 1
0.914 36
0.5 2
Advice Dept. Planning and Management (complete) 42
Drafting fee refund letter (complete) 2
0.96 25
B & I request original drawing (complete) 26
0.875 24
0.909 11
Processing advice (complete) 30
0.833 29
0.5 1
0.952 29
Authorize request (complete) 31
Sending invoice for driveway construction (complete) 1
Compose and send Refusal (complete) 11
0.5 1
0.5 1
0.96 30
0.667 2
0.9 9
Contract construction entrance div BUI (complete) 1
Publication decision (complete) 31
Send fee refund (complete) 3
0.5 1
0.667 30
0.667 2
End procedure (complete) 43

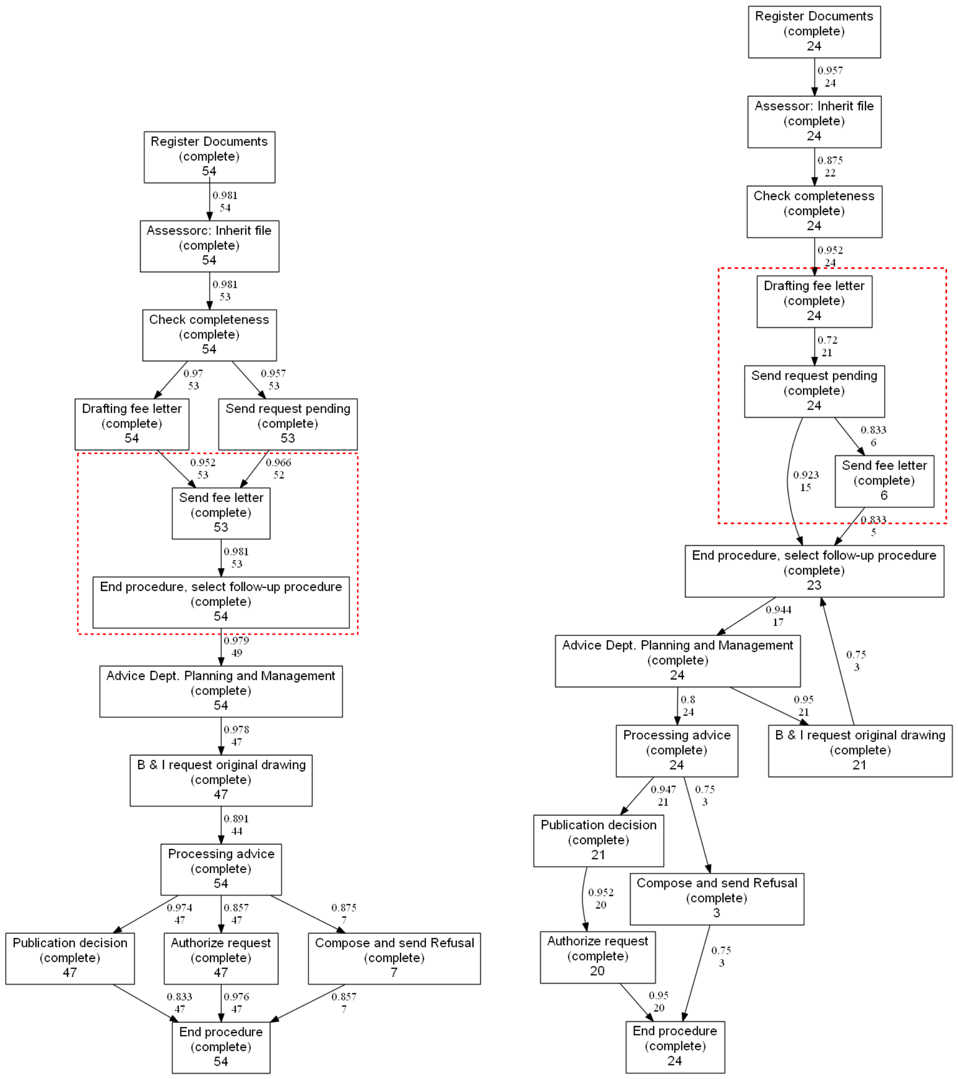
Figure 11.33: Heuristic net of the permit process for driveway construction discovered using the event log $\mathcal{L}_4$. Refunds are done only for selected cases in case of permit refusal and the number of contracts given to the BUI division has reduced drastically when compared to the model in Figure 11.32.


Register Documents (complete) 45
Assessor: Inherit file (complete) 45
Check completeness (complete) 45
Send request pending (complete) 45
Drafting fee letter (complete) 45
Send fee letter (complete) 45
Advice Dept. Planning and Management (complete) 45
End procedure, select follow-up procedure (complete) 45
B & I request original drawing (complete) 37
Processing advice (complete) 43
Compose and send Refusal (complete) 6
Publication decision (complete) 39
Authorize request (complete) 39
Send fee refund (complete) 1
End procedure (complete) 45
Register Documents (complete) 24
Assessor: Inherit file (complete) 24
Check completeness (complete) 24
Send request pending (complete) 24
Drafting fee letter (complete) 24
Send fee letter (complete) 24
End procedure, select follow-up procedure (complete) 24
Advice Dept. Planning and Management (complete) 24
B & I request original drawing (complete) 14
Processing advice (complete) 24
Authorize request (complete) 21
Publication decision (complete) 20
Compose and send Refusal (complete) 2
End procedure (complete) 24

(a) event log $\mathcal{L}_5$ (b) event log $\mathcal{L}_6$

Figure 11.34: Heuristic nets of the permit process for driveway construction discovered using the event logs $\mathcal{L}_5$ and $\mathcal{L}_6$. The marked regions signify the regions where changes are perceived when compared to the previous model. In the model in (a), the activities Advice Dept. Planning and Management and B & I request original drawing happen more often in parallel with Send fee letter and End procedure, select follow-up procedure, unlike the model in Figure 11.33. Furthermore, the municipality no longer assigns contracts to the BUI division. The activities Drafting fee letter and Send request pending can be executed in parallel in (b) when compared to the model in (a).

procedure. Figure 11.35(b) depicts the process model discovered using the Heuristics miner on the event log $\mathcal{L}_8$. The change in this model pertains to the activity Send fee letter. Unlike all of the previous models, the fee letter is sent only to some of the applicants (for just 6 of the 24 permit requests was a fee letter sent). Due to this, the probability of activities differ in the traces in $\mathcal{L}_8$ when compared to the traces in $\mathcal{L}_7$. The J-measure elegantly captures this and reflects this as a change in behavior.


Register Documents (complete) 54
Assessorc: Inherit file (complete) 54
Check completeness (complete) 54
Drafting fee letter (complete) 54
Send request pending (complete) 53
Send fee letter (complete) 53
End procedure, select follow-up procedure (complete) 54
Advice Dept. Planning and Management (complete) 54
B & I request original drawing (complete) 47
Processing advice (complete) 54
Publication decision (complete) 47
Authorize request (complete) 47
Compose and send Refusal (complete) 7
End procedure (complete) 54
Register Documents (complete) 24
Assessor: Inherit file (complete) 24
Check completeness (complete) 24
Drafting fee letter (complete) 24
Send request pending (complete) 24
Send fee letter (complete) 6
End procedure, select follow-up procedure (complete) 23
Advice Dept. Planning and Management (complete) 24
B & I request original drawing (complete) 21
Processing advice (complete) 24
Publication decision (complete) 21
Compose and send Refusal (complete) 3
Authorize request (complete) 20
End procedure (complete) 24

(a) event log $\mathcal{L}_7$ (b) event log $\mathcal{L}_8$

Figure 11.35: Heuristic nets of the permit process for driveway construction discovered using the event logs $\mathcal{L}_7$ and $\mathcal{L}_8$. The sending of fee letter has to happen before the activity End procedure, select follow-up procedure in (a) while the activity is optional in (b).

To summarize, the features and the framework proposed in this thesis for handling concept drifts show significant promise in detecting behavioral changes by analyzing event logs. The detection of such change points can help us put the results of process mining in a right perspective and enables an organization to take appropriate measures when a change in behavior is perceived. Furthermore, the empirical studies on the municipality event logs showed that the framework is robust. It is able to detect changes in real-life event logs even with a small number of cases.

11.3 Workflow of Field Service Engineers

Our third case study was done in close collaboration with a global leader in professional and consumer healthcare. As mentioned in Chapter 9, whenever a customer (hospital) complains about a malfunction of an X-ray machine, a Field Service Engineer (FSE) is despatched (if needed) to fix the problem. Depending on the type of the problem, the field service process can consist of the corrective actions such as *configuration*, *calibration*, and/or *part replacements*. Due to the complexity of system architecture and the flexibility of system use, the diagnostic procedures are not always easy, e.g., for some problems, we notice a high variation in the time to repair. This implies that some of the cases are not easy to diagnose and efforts are needed to help FSEs. The actions performed by an FSE during diagnosis (maintenance) are expected to be manifested in event logs. The organization is interested in understanding the workflow of FSEs and in identifying the bottlenecks in the process (e.g., discover which activities take more time). Furthermore, they are also interested in assessing whether all of the mandatory activities such as calibration of the system after a part replacement are performed. Such insights can help them in improving their current diagnostic procedures, customer service quality, and more importantly, customer satisfaction besides reducing costs.

11.3.1 Data Selection and Preparation

In Chapter 9, we already explained the logging capabilities of X-ray systems and their event logs in detail. Without repeating the description provided there, we present a high-level view of the data selection and preparation process for the analysis of FSE workflows in this section.

Data Selection

As mentioned in Chapter 9, there are two data sources, the system event logs and job sheets. The data selection process for the analysis of workflow of FSEs during a part replacement is similar to that of the data selection process for signature discovery explained in Chapter 9. For example, if we are interested in understanding the workflow of FSEs while replacing part A, we identify all systems from the job sheets database where that part had been replaced. Each call in the job sheets database with this part replacement is associated with a call open date and a call close date. For each call, we consider logs from the corresponding system between the call open and the call close date (both days inclusive). Note that this differs slightly from the time window used for signature discovery (cf. Section 9.3.2), where we considered logs from a few days before/after the call open/close date. Since FSEs take charge only after a complaint has been raised, it is reasonable to consider logs only on/after the call open date.

Defining Cases: Scoping and Filtering

As mentioned in Chapter 9, there are different modes during the operation of an X-ray machine, viz., start-up (SU), shutdown (SD), shutdown completed (SDC), normal operation (NO), warm-restart (WR), and field service (FS). Events are logged in all of these system modes. The start-up, shutdown, shutdown completed, and normal operation modes are common and almost occur everyday during regular usage of the system. Operations meant for regular use by the clinicians are logged under the normal operation mode. The field service system mode mostly manifests during system maintenance (operations specific to FSEs are logged under this mode) while the warm-restart system mode refers to an automatic restart of the system and occurs rarely. The field service system mode is similar to the administrator mode in an operating system. FSEs can also operate the system in the normal operation mode. During a single day, the system could have been started or shutdown multiple times. The shutdown completed system mode occurs immediately after every shutdown mode. The sequence of events in any particular mode can be considered as a *session* of system's operation. Figure 11.36 depicts a scenario of system logs between the call open date and the call close date where the events are marked by sessions.

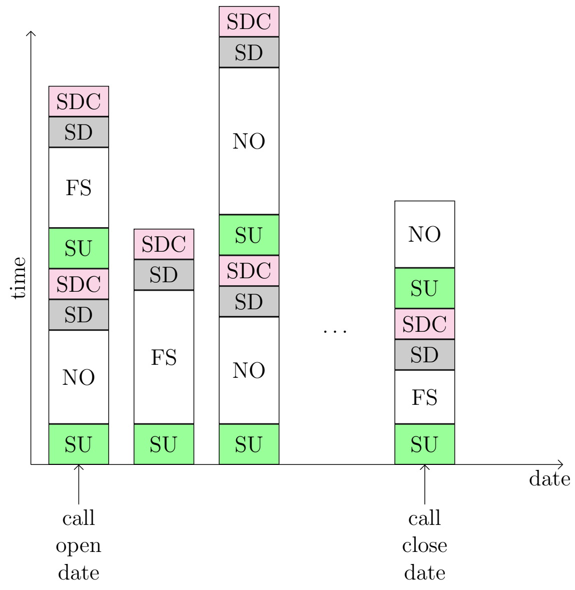


Figure 11.36: Events in a system log marked as sessions. The continuous sequence of events in any particular system mode constitutes a session of system's operation. SU:start-up mode, NO:normal operation, SD:shutdown mode, SDC:shutdown completed mode, FS:field service mode. In this example, the warm-restart system mode is not manifested.

For understanding the workflow of FSEs during part replacements, we consider sessions between the first and the last field service sessions (both inclusive) and *create one process instance (case) per call by juxtaposing the sessions.* The domain experts are mostly interested in understanding the operations of FSEs during the field service modes. Nonetheless, the activities during the normal operation mode are also important. However, we are not interested in the events during the startup, shutdown, and shutdown completed modes (as these are standard operations that are predefined (similar to computer boot up)). Therefore, we record just two events for each of the startup, shutdown, and shutdown completed sessions signifying the start and completion of the session. This will be handy during performance analysis via log replay. Similarly, since we are not interested in the low level details of the normal operation at first glance, we create an abstract event for each normal operation session and create a sub-log for the normal operations where a process instance corresponds to a session of normal operation. We set the start time of the abstract event to the timestamp of the first event in the session and the completion time of the abstract event to the timestamp of the last event in the session. Figure 11.37 depicts the creation of a case for each call.

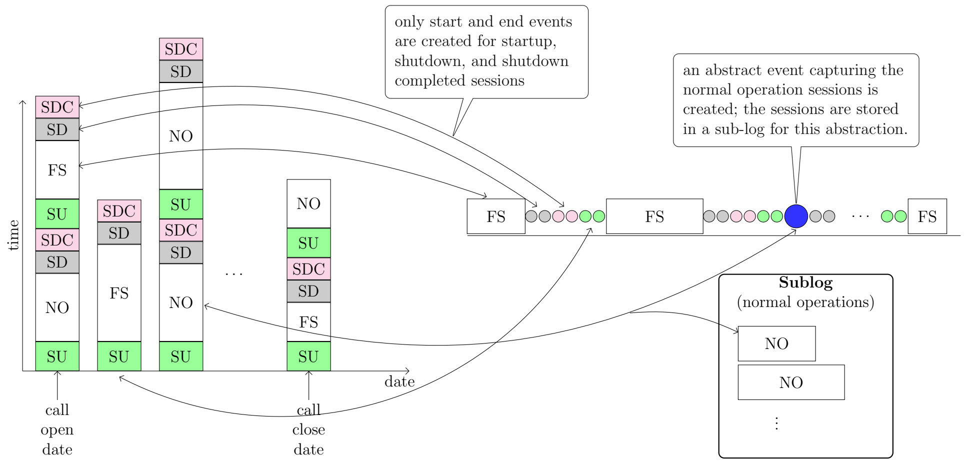

only start and end events are created for startup, shutdown, and shutdown completed sessions
an abstract event capturing the normal operation sessions is created; the sessions are stored in a sub-log for this abstraction.
time
date
call open date
call close date
SDC
SD
FS
SU
NO
Sublog (normal operations)

Figure 11.37: Defining a process instance (case) per call by juxtaposing the sessions. Normal operation sessions are abstracted and a sub-log is created with process instances corresponding to the sessions.

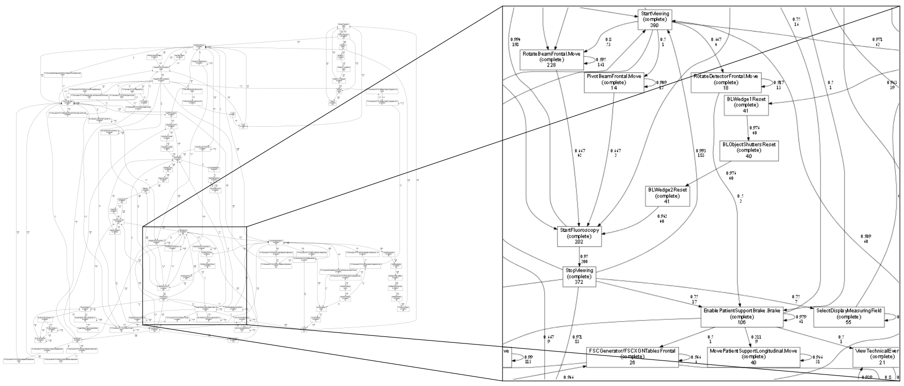
In scenarios where we do not want to consider the events during normal operation mode, we can follow the same principle as in startup/shutdown modes, i.e., create two events signifying the start/completion of normal operation mode instead of an abstract event. One might want to analyze the normal operation mode separately by creating a log with the sessions corresponding to the normal operation mode as process instances.

11.3.2 Discovering Hierarchical Workflows

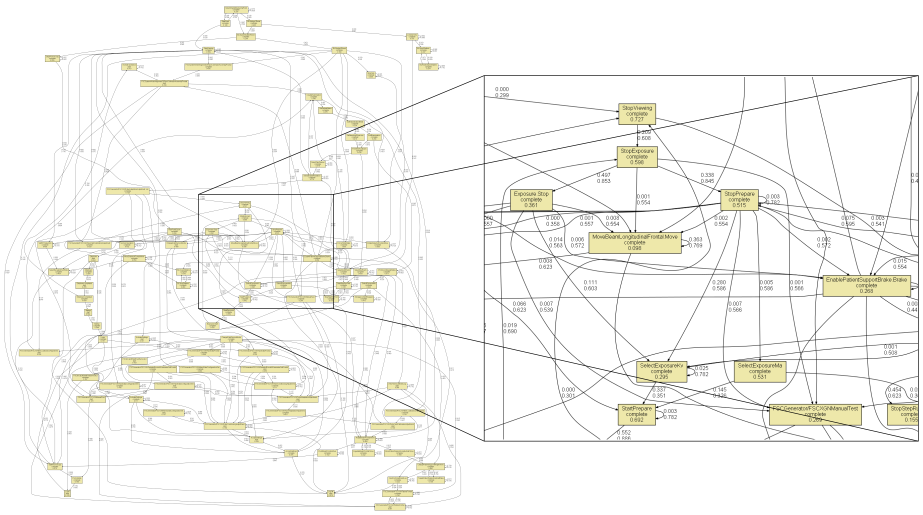
In this section, we report on the discovery of workflow capturing the activities of FSEs during one of the part replacements in an X-ray machine. The chosen part is one of the most expensive parts in an X-ray machine and has a large deviation in MTTR. We considered all systems of a particular software version where this part had been replaced in the past. We created an event log from the system logs based on the procedure defined in the previous section. In this experiment we confine ourselves to only the activities performed by FSEs in the field service mode, i.e., we represent each session pertaining to system modes other than field service with two events signifying the start and completion of the session without bothering about the actual events in those sessions. However, all events executed under the field service mode are considered for analysis. The event log contains 16 cases and 8,867 events distributed over 192 activities. Significant number of activities happen very rarely (their relative frequency of occurrence is less than 0.05%). We filter (remove) such activities. The filtered event log contains 16 cases and 8,677 events referring to 97 activities. Note that we see a reduction from 192 to 97 activities. Figures 11.38(a) and (b) depict the process models uncovered on this log using the Heuristics miner [264] and the classical Fuzzy miner [94] respectively. The figures clearly expose the complexity of this event log and the limitations of existing discovery algorithms. The fine-granularity of events is one of the primary reasons for this observed behavior. The uncovered process models are totally incomprehensible. Domain experts are interested in understanding the workflow of FSEs at the level of procedures that engineers execute rather than at the (fine-granular) level of system commands as manifested in the event log.

As discussed in Chapters 3 and 6, fine-granular event logs can be dealt by defining abstractions and using our two-phase approach to process discovery. We first consider the tandem arrays and maximal repeat patterns in the event log and define abstractions over them as discussed in Chapter 3. Table 11.3 depicts a few pattern alphabets and their corresponding abstractions. The uncovered common execution patterns are found to have strong domain significance. For example, consider the pattern alphabet 3 in Table 11.3. Before scanning an image of a patient, the details about the patient and the examination type (e.g., left ventricle scan) need to be entered on an X-ray machine for bookkeeping purposes. This functionality is captured as a common execution pattern in the event log. The chosen part (for analysis) is one of the critical components of an X-ray machine and if it is broken, the functionality of the system as a whole is compromised (image scans are no longer possible). This pattern reflects the behavior of FSEs attempting to record image scans as part of their system diagnosis procedure. After defining the abstractions, we transform the input event log into higher-levels of abstraction using the approach presented in Section 6.3. This procedure generates a sub-log for each abstract activity, which can be used to mine the subprocesses captured by the abstract activity. The transformed event log contains 16 cases and 2154 events referring to 29 activities.

Figure 11.39 depicts the top-level process model discovered using the Fuzzy Map



(a) Heuristic net



(b) Classical Fuzzy model

Figure 11.38: Process models discovered using the Heuristics miner and the classical Fuzzy miner on the event log capturing the activities in the field service mode.

miner on the transformed log. The blue (dark) colored nodes are abstract nodes that can be zoomed into. Note that this model is much more simpler and easily comprehensible when compared to models discovered using classical approaches as in Figure 11.38. We can see that certain tasks during maintenance such as calibration, configuration, and adjustments captured as abstract activities. Figure 11.40 depicts the subprocesses corresponding to four of the abstract nodes. For example, the events capturing the start/completion of the startup/shutdown and normal operation modes are captured using an abstract activity, the subprocess of which is as depicted in Figure 11.40(a). FSEs typically inspect the system logs to see if there are any (known) error patterns. This is reflected in the activities View Technical Event Log

Table 11.3: Pattern alphabets and abstractions defined using tandem arrays and maximal repeat patterns on the event log.

S.No	Pattern Alphabet	Abstraction
1	{Start Fluoroscopy, Stop Fluoroscopy, Start Viewing, Stop Viewing, Lower Priority, Minimize}	Fluoroscopy/Exposure and Viewing
2	{Start Exposure, Stop Exposure, Start Viewing, Stop Viewing, Start Prepare, Stop Prepare, Lower Priority, Minimize}	
3	{Create Examination, Add Examination, Set Patient Name, Set Patient Date of Birth, Set Attending Physician}	Examination Preparation and Administration
...	...	...

in the subprocess for System Information depicted in Figure 11.40(c). By simplifying event logs with abstractions, we are able to achieve meaningful and comprehensible process models. We next look at the workflow of FSEs during the normal operation mode.

Normal Operation Workflow

We consider the sessions defining the normal operation mode during the diagnosis of a part replacement and analyze the activities performed by FSEs. The event log contains 113 cases (each corresponding to one normal operation sessions) and 77,755 events distributed over 182 activities. Significant number of these activities occur very rarely. We filter all activities whose relative occurrence frequency is less than 0.05%. The filtered event log contains 113 cases and 76,754 events referring to 92 activities. Figures 11.41(a) and (b) depict the process models uncovered on this log using the Heuristics miner and the classical Fuzzy miner respectively. Again, we see incomprehensible spaghetti-like models. We uncover common execution patterns, define abstractions, and use our two-phase approach to alleviate this problem. The transformed log (with abstractions) contains 113 cases and 10,387 events referring to 20 activities.

Figure 11.42 depicts the top-level process model discovered using the Fuzzy Map miner on this transformed log. We can see that the model discovered using our two-phase approach is simpler and more comprehensible. Further note that at the beginning of the process, there are two activities FSA startup and Change Field Service Mode. These activities signify that the normal operation mode is being executed with field service privileges (this is analogous to the sudo program on unix-like operating systems). Figure 11.43(a) depicts the subprocess for the abstract activity Fluoroscopy, Exposure, and Viewing while Figure 11.43(b) depicts the subprocess for the abstract activity Beam Limitation. The beam limitation subprocess is concerned with adjusting the wedges and shutters in an X-ray machine to direct X-ray beams to the desired regions of scan. There are two wedges (Wedge1 and Wedge2) in the X-ray machine and the permissible operations on them are translation (movement) inwards/outwards or stop and rotation. This is elegantly captured in the Beam

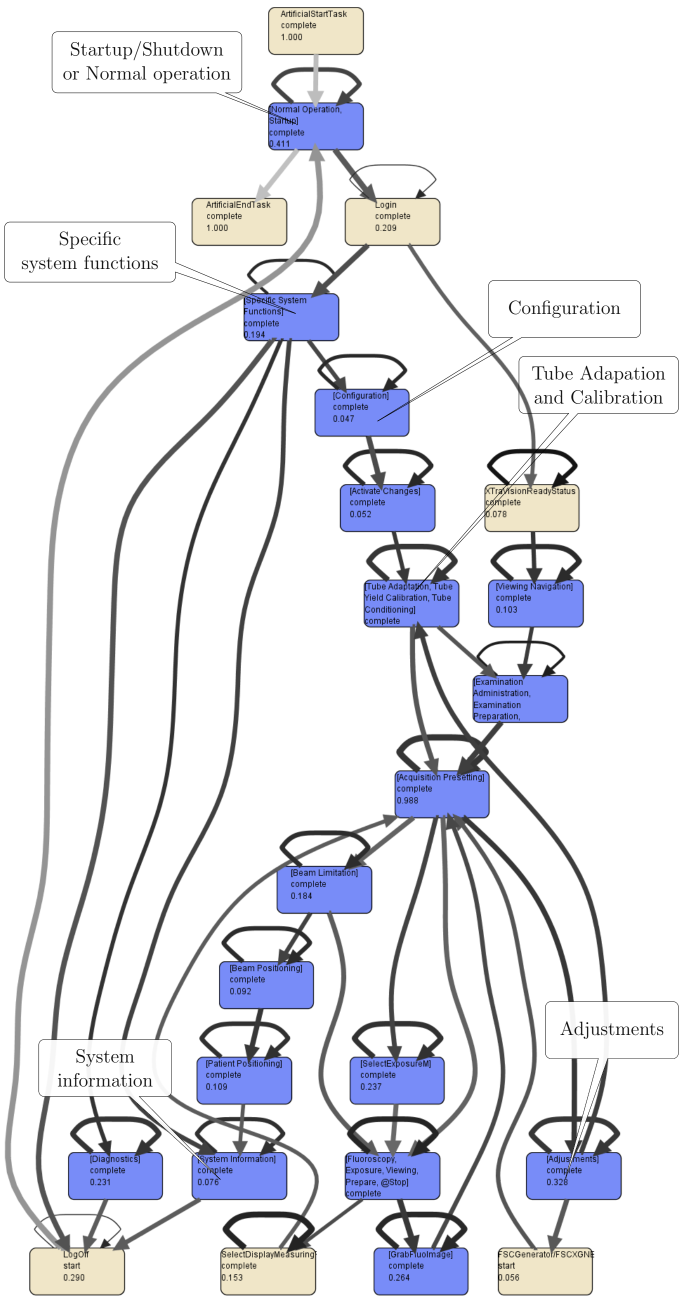
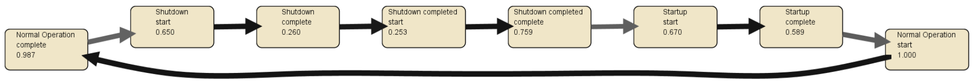
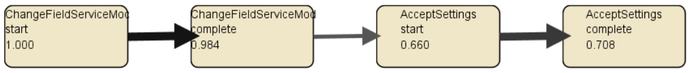

ArtificialStartTask complete 1.000
Startup/Shutdown or Normal operation
[Normal Operation, Startup] complete 0.411
ArtificialEndTask complete 1.000
Login complete 0.209
Specific system functions
[Specific System Functions] complete 0.194
Configuration
[Configuration] complete 0.047
Tube Adapation and Calibration
[Activate Changes] complete 0.052
XTraVisionReadyStatus complete 0.078
[Tube Adaptation, Tube Yield Calibration, Tube Conditioning] complete
[Viewing Navigation] complete 0.103
[Examination Administration, Examination Preparation,
[Acquisition Presetting] complete 0.988
[Beam Limitation] complete 0.184
[Beam Positioning] complete 0.092
[Patient Positioning] complete 0.109
[SelectExposureM] complete 0.237
Adjustments
System information
[Diagnostics] complete 0.231
[System Information] complete 0.076
[Fluoroscopy, Exposure, Viewing, Prepare, @Stop] complete
[Adjustments] complete 0.328
LogOff start 0.290
SelectDisplayMeasuring complete 0.153
[GrabFluoImage] complete 0.264
FSCGenerator/FSCXGNE start 0.056

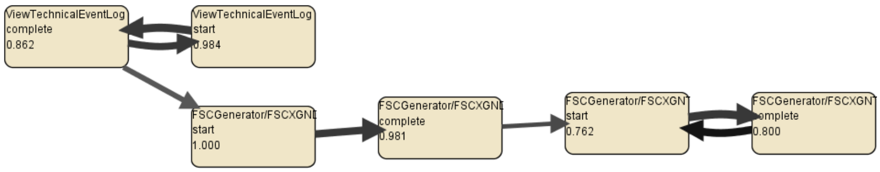
Figure 11.39: The top-level process model corresponding to the activities performed by FSEs in the field service system mode.



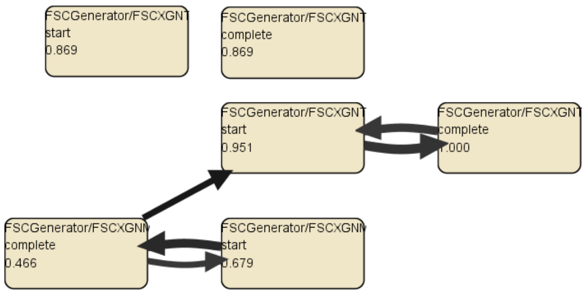
(a) Startup/Shutdown and/or Normal Operation



(b) System Specific Functions



(c) System Information

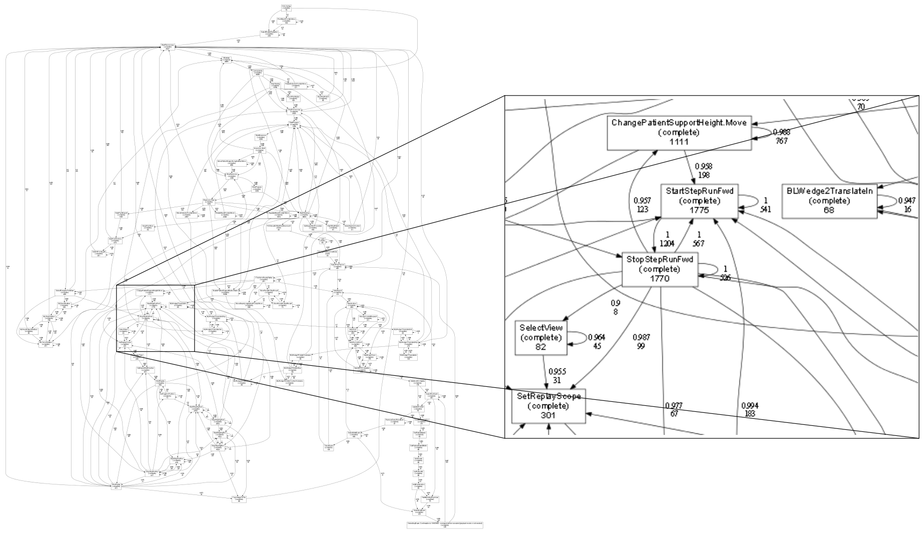


(d) Configuration

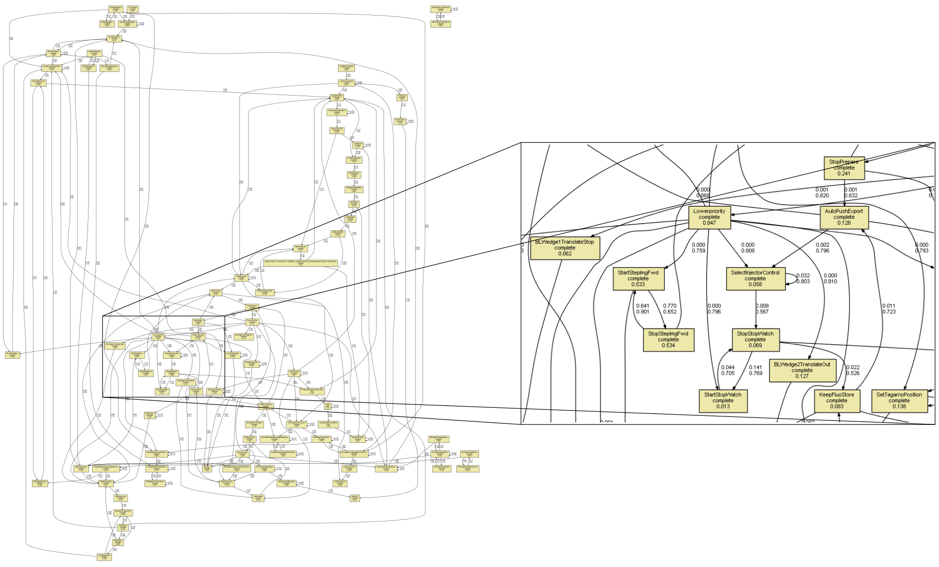
Figure 11.40: The subprocesses corresponding to the abstract activities Startup/Shutdown and/or Normal Operation, System Specific Functions, System Information, and Configuration.

Limitation abstraction. Similarly, the shutters can be opened/closed, stopped, or reset.

We have applied this methodology to discover the workflow of FSEs for several other part replacements. In all of the cases, we were able to obtain meaningful and comprehensible process maps. Based on these experiences, we summarize that the approaches presented in this thesis of forming abstractions by exploiting common execution patterns in event logs, the preprocessing of logs with these abstractions, in conjunction with the two-phase approach for process discovery enables the discovery of comprehensible hierarchical process models (a.k.a. process maps) at desired levels of granularity. In the next section, we attempt at enriching these mined models with performance information so that bottlenecks can be easily identified.


ChangePatientSupportHeight.Move (complete) 1111
StartStepRunFwd (complete) 1775
BLWedge2TranslateIn (complete) 68
StopStepRunFwd (complete) 1770
SelectView (complete) 82
SetReplayScope (complete) 301

(a) Heuristic net


StopPrepare complete 0.241
Lowerpriority complete 0.847
AutoPushExport complete 0.126
BLWedge1TranslateStop complete 0.062
StartStepImgFwd complete 0.533
SelectInjectorControl complete 0.058
StopStepImgFwd complete 0.534
StopStopWatch complete 0.069
BLWedge2TranslateOut complete 0.127
StartStopWatch complete 0.013
KeepFluoStore complete 0.083
SetTagarnoPosition complete 0.138

(b) Classical Fuzzy model

Figure 11.41: Process models discovered using the Heuristics miner and the classical Fuzzy miner on the event log defined by the sessions of normal operation mode.

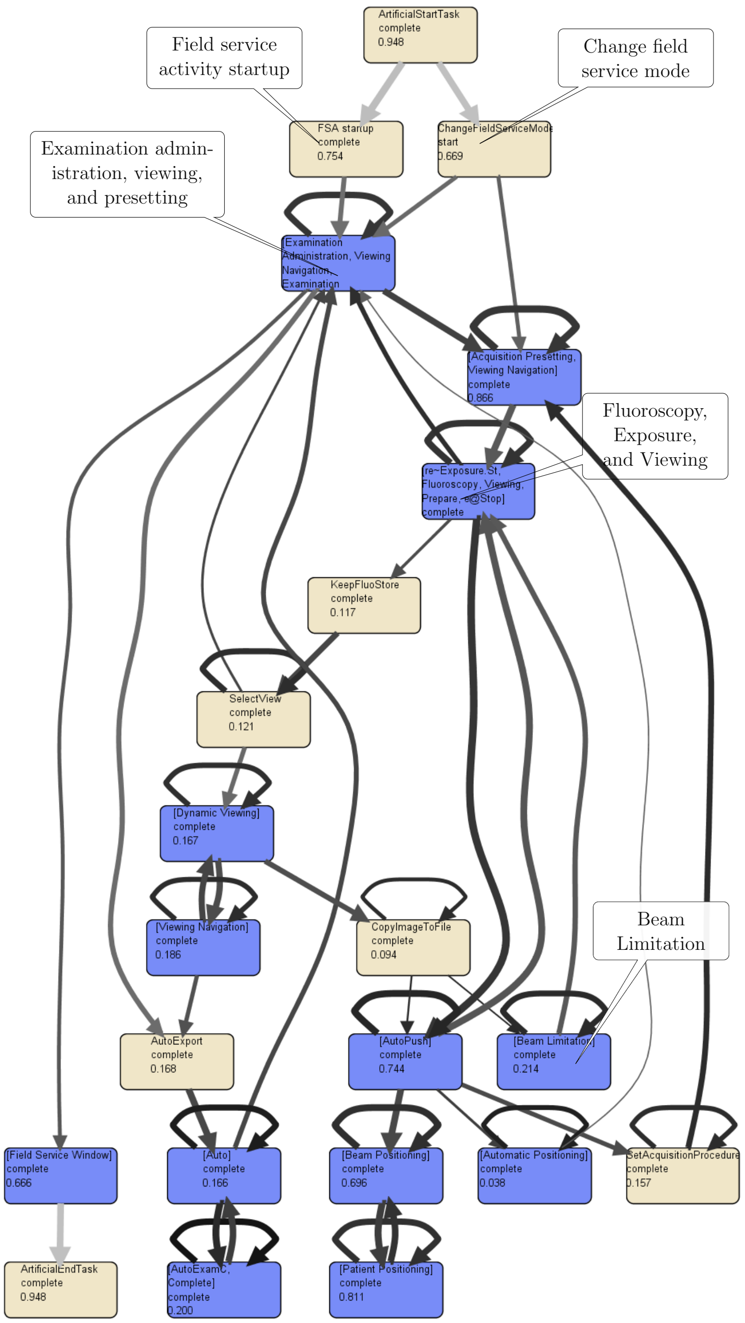
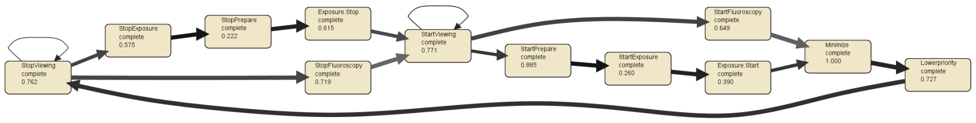
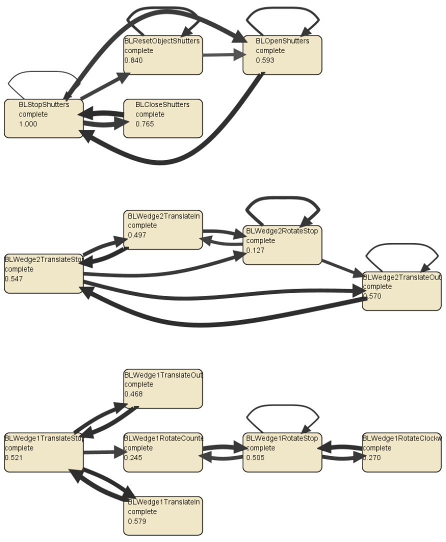

ArtificialStartTask complete 0.948
Field service activity startup
Change field service mode
FSA startup complete 0.754
ChangeFieldServiceMode start 0.669
Examination administration, viewing, and presetting
[Examination Administration, Viewing Navigation, Examination
[Acquisition Presetting, Viewing Navigation] complete 0.866
Fluoroscopy, Exposure, and Viewing
[re~Exposure.St, Fluoroscopy, Viewing, Prepare, e@Stop] complete
KeepFluoStore complete 0.117
SelectView complete 0.121
[Dynamic Viewing] complete 0.167
[Viewing Navigation] complete 0.186
CopyImageToFile complete 0.094
Beam Limitation
AutoExport complete 0.168
[AutoPush] complete 0.744
[Beam Limitation] complete 0.214
[Field Service Window] complete 0.666
[Auto] complete 0.166
[Beam Positioning] complete 0.696
[Automatic Positioning] complete 0.038
SetAcquisitionProcedure complete 0.157
ArtificialEndTask complete 0.948
[AutoExamC, Complete] complete 0.200
[Patient Positioning] complete 0.811

Figure 11.42: The top-level process model corresponding to the sessions operated in the normal operation system mode.



(a) Fluoroscopy, Exposure and Viewing



(b) Beam Limitation

Figure 11.43: The subprocesses corresponding to the abstract activities Fluoroscopy, Exposure and Viewing and Beam Limitation.

11.3.3 Performance Analysis

One means of identifying the hotspots (bottlenecks) in a process is to replay an event log on a process model. During replay relevant performance measures can be captured. We considered the process maps discovered in the previous section and replayed their respective input logs (the logs from which the models are discovered) using the replay technique presented in Chapter 7. Figure 11.44 depicts the annotated Fuzzy map for the workflow of FSEs considering the activities executed in the field service mode. We used the level average thresholds for coloring the nodes with bounds of 0.3. In other words, if a node/edge's average throughput time is less than 0.7 of the average node/edge throughput time of the process model at this level, then the node/edge is colored green. If its value is between 0.7 and 1.3, then it is colored yellow and if its value is greater than 1.3, it is colored red. For example, in Figure 11.44, we see the abstract activity Startup, Shutdown, and Normal operation colored red. This is expected because the FSE performs a lot of operations in the normal operation mode. Figure 11.45(a) depicts the subprocess corresponding to this abstract activity. We can get further insights on where the contributing factors (to large execution times) are by analyzing this subprocess. In this subprocess, we have 8 events that correspond to the start and completion events of sessions in four system modes: startup, shutdown, shutdown completed, and normal operation. Among these four, the most variable component is the normal operation mode because the activities performed vary depending on the need of any operating day of a hospital. Accordingly, we see the edge Normal Operation-start → Normal operation-complete colored red. Normally, the events captured during the startup mode are fixed with very little variability. These typically involve handshaking commands with all the components/parts of the system (analogous to operating system booting process). The time it takes for the system to wake up is negligible compared to the time spent in normal operation mode. Thus, we see the edge between Startup-start → Startup-complete colored green.

Figure 11.45(b) depicts the annotated subprocess corresponding to the abstract activity System Information. The edge connecting View Technical Event Log-start and View Technical Event Log-complete is colored red. This activity corresponds to the FSE manually inspecting the system logs to check for the existence of any known patterns. This is a cumbersome process and this is one of the motivating factors for the need for automated discovery of signature patterns, which we addressed in Chapter 9. In this fashion, one can try to analyze the activities and flows in the process that are time consuming and if possible remedial actions can be taken such as optimizing the functionality.

Figure 11.46 depicts the annotated Fuzzy map corresponding to the workflow of FSEs in normal operation mode using the same coloring strategy as earlier. Figures 11.47(a) and (b) depict the annotated Fuzzy map for two of the subprocesses, viz., Beam Limitation and Field service window in the top-level process model. Let us consider the subprocess in Figure 11.47(a). As mentioned earlier, the wedges in an X-ray machine can be moved and/or rotated. The rotation of wedges is more com-

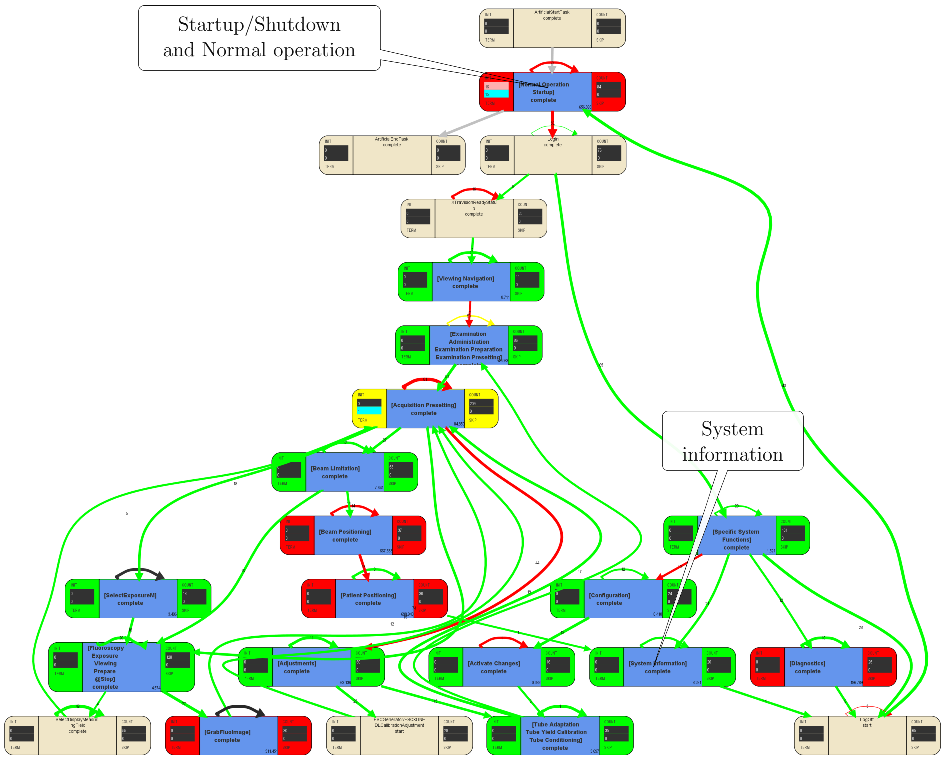
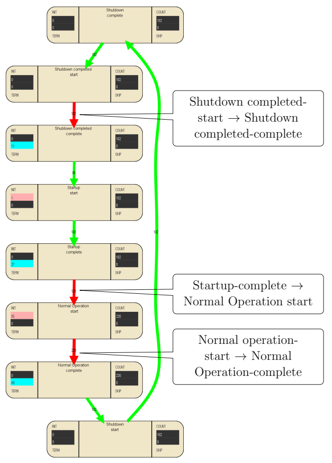

Startup/Shutdown and Normal operation
ArtificialStartTask complete
[Normal Operation Startup] complete
ArtificialEndTask complete
Login complete
XTraVisionReadyStatus complete
[Viewing Navigation] complete
[Examination Administration Examination Preparation Examination Presetting] complete
[Acquisition Presetting] complete
[Beam Limitation] complete
[Beam Positioning] complete
System information
[Specific System Functions] complete
[SelectExposureM] complete
[Patient Positioning] complete
[Configuration] complete
[Fluoroscopy Exposure Viewing Prepare @Stop] complete
[Adjustments] complete
[Activate Changes] complete
[System Information] complete
[Diagnostics] complete
SelectDisplayMeasuringField complete
[GrabFluoImage] complete
FSOGenerator/FSCXDNEDLCalibrationAdjustment start
[Tube Adaptation Tube Yield Calibration Tube Conditioning] complete
LogOff start

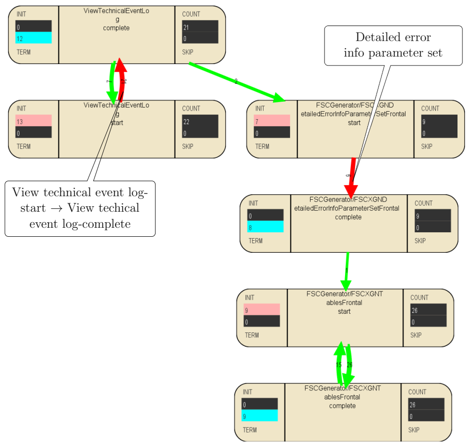
Figure 11.44: The top-level process model, of the activities performed by FSEs in the field service mode, showing performance related information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±30% with the average node throughput time of 119.034 units and an average edge throughput time of 77.198 units.

plex than the lateral/vertical movements. This is reflected in the edges connecting to/from the activity corresponding to the rotation of wedges. These edges acquire a red color because they are relatively more time consuming.

In this fashion, one can explore annotated process models to identify potential bottlenecks in a process.


Shutdown completed-start → Shutdown completed-complete
Startup-complete → Normal Operation start
Normal operation-start → Normal Operation-complete

(a) Startup/shutdown and/or normal operation


ViewTechnicalEventLog complete
Detailed error info parameter set
View technical event log-start → View techical event log-complete
FSCGenerator/FSCXGNDetailedErrorInfoParameterSetFrontal start
FSCGenerator/FSCXGNTablesFrontal start
FSCGenerator/FSCXGNTablesFrontal complete

(b) System information

Figure 11.45: The subprocesses corresponding to Startup/shutdown and/or normal operation and System information annotated with performance information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±30%. The average edge throughput time of Startup/shutdown and/or normal operation is 58.020 units while that of System information is 2.298 units.

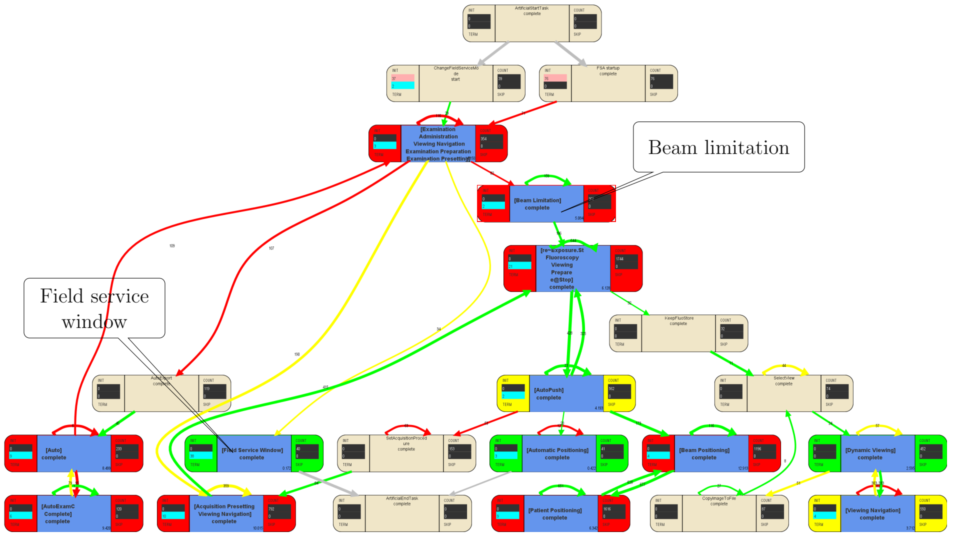
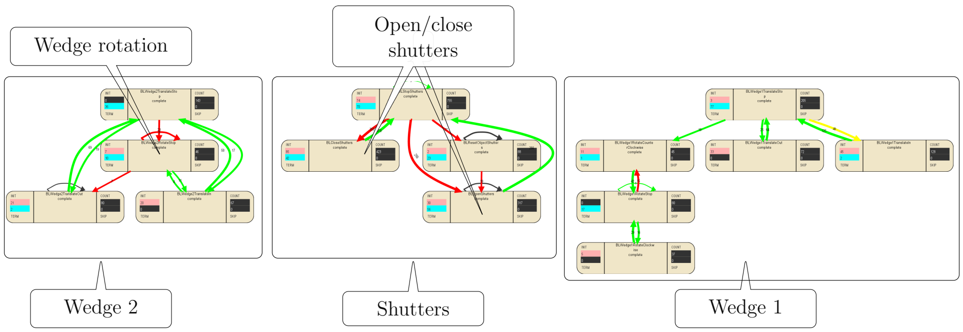
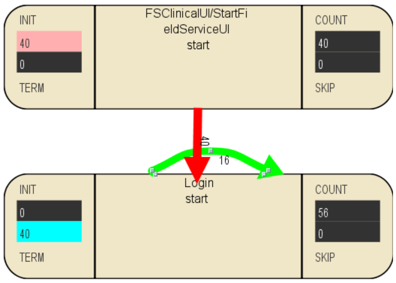

Beam limitation
Field service window
ArtificialStartTask complete
[Beam Limitation] complete
[AutoPush] complete
[Auto] complete
[Field Service Window] complete
[Automatic Positioning] complete
[Beam Positioning] complete
[Dynamic Viewing] complete
[Acquisition Presetting Viewing Navigation] complete
ArtificialEndTask complete
[Patient Positioning] complete
[Viewing Navigation] complete

Figure 11.46: The top-level process model of the sessions corresponding to the normal operation mode showing performance related information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±30%. The average node throughput time is 40.005 units and the average edge throughput time is 13.576 units.



(a) Beam limitation



(b) Field service window

Figure 11.47: The subprocesses corresponding to Beam Limitation and Field Service Window annotated with performance information. The coloring of nodes/edges is based on model-specific averages with tolerance limits of ±30%. The average edge throughput time for Beam limitation is 10.220 units while that of the Field service window is 0.91 units.

11.4 Conclusions

In this chapter, we have presented three case studies that have been performed in the context of the work presented in this thesis. Each of these case studies exhibits one or more of the characteristics of large scale event logs addressed in this thesis. These case studies substantiate our claim that preprocessing, leading to simplified event logs, based on the perspective of analysis is crucial in gaining meaningful insights.

Chapter 12
Conclusions

To conclude this thesis, we first summarize some of the motivating factors that has driven the course of the work presented in this thesis followed by our contributions. Although the techniques and concepts presented in this thesis take a step forward in addressing some of these factors, several challenges still remain to be addressed to improve the applicability of the techniques presented in this thesis, and the applicability of process mining in general. We list some of these challenges and provide some directions for future work.

12.1 Summary of Contributions

Process mining is an important tool for organizations managing nontrivial operational processes. During the last five years, process mining has expanded its reach to new domains and applications, atypical of workflow-like information systems, e.g., analyzing event logs from high-tech systems such as X-ray machines, wafer scanners, copiers and printers, and mission critical information systems. These applications bring along with it new sets of challenges, e.g., fine-granular events, heterogeneity in event logs, voluminous data, unreliable timestamps, etc. The challenges will only be further inclined to grow what with the organizations predicted to churn out burgeoning volumes of transactional data, via all means from networked sensors to social media, referred to as the "big data" [147].

In *Part* I, we presented some of the characteristics of large scale event logs and three major challenges for process mining, viz., dealing with less-structured processes, dealing with process changes, and provisions for process diagnostics. We highlighted the issues with current techniques and the need for advancements in process mining. We presented some of the diagnostic questions that business analysts are interested in obtaining answers to. Chapter 2 introduced several notations and concepts in the context of process mining such as event logs and process modeling formalisms and those needed for explaining the concepts in the thesis.

In *Part* II, we hypothesized that simplification of event logs can improve process mining results. Therefore, we focused on techniques for preprocessing event logs. First, we presented an approach of forming abstractions of fine-granular events by exploiting some common execution patterns, manifested as sequences of activities, in event logs (Chapter 3). We established relationships between these patterns and some of the process modeling constructs. By using these abstractions, low-level events can be replaced with high-level abstract activities and process mining be applied on these abstracted logs. Next, we proposed techniques for partitioning event logs into homogenous subsets as a means of dealing with heterogeneity in event logs (Chapter 4). We highlighted the importance of considering contexts of execution in forming homogenous partitions and showed that process mining results can be improved by analyzing these subsets independently when compared to analyzing the whole event log. In Chapter 5, we focused on techniques for dealing with process changes. We presented techniques for determining the time points when changes took place and the regions where processes have changed. Analyzing such changes is of utmost importance to get an accurate insight on process executions at any instant of time.

In *Part* III, we presented several advancements in topics in process mining. First, we focused on one of the challenging tasks in process mining, i.e., dealing with less-structured processes. We presented a two-phase approach to process discovery in Chapter 6. We presented an event log transformation approach that exploits the abstractions defined over common execution patterns presented in Chapter 3. The transformation of logs constitute the first phase of our two-phase approach. During this transformation, sub-logs, capturing the manifestation of activities captured under an abstraction, are created for each abstract activity. The second phase deals with mining process maps. Abstract activities can be zoomed in to view the subprocesses captured underneath them. Chapters 7–9 dealt with techniques for process diagnostics. We presented an approach of replaying an event log onto process maps and gathering relevant performance measures (Chapter 7). These measures can be annotated on a process map to identify bottlenecks. We presented an approach of aligning traces to answer a variety of diagnostic questions (Chapter 8). Trace alignment extends the scope of conformance checking in process mining by finding deviations by analyzing the raw traces without the need for process models. Chapter 9 presented an approach of finding symptomatic signature patterns that explain the behavior of different classes of traces in an event log. Such signatures can be used for fault diagnosis and fraud detection.

In this last part (*Part* IV), we presented the tool support, realized as packages in the ProM framework, for the concepts presented in this thesis (Chapter 10). We presented three case studies to demonstrate the applicability of the techniques developed in this work (Chapter 11). In this chapter, we take a step back and reflect on the material presented in the previous chapters and sketch directions for future work.

Figure 12.1 summarizes the contributions of this thesis. We have presented several features with multitude of applications, e.g., common execution patterns such as tandem arrays and maximal repeats catering to abstractions of events, trace clustering, and signature discovery; causal foot print based features catering to concept drifts, etc. These constitute the class of feature extraction techniques. We have presented feature selection techniques to filter/prune features that are insignificant or less significant, e.g., the various pattern count metrics. Based on these features,

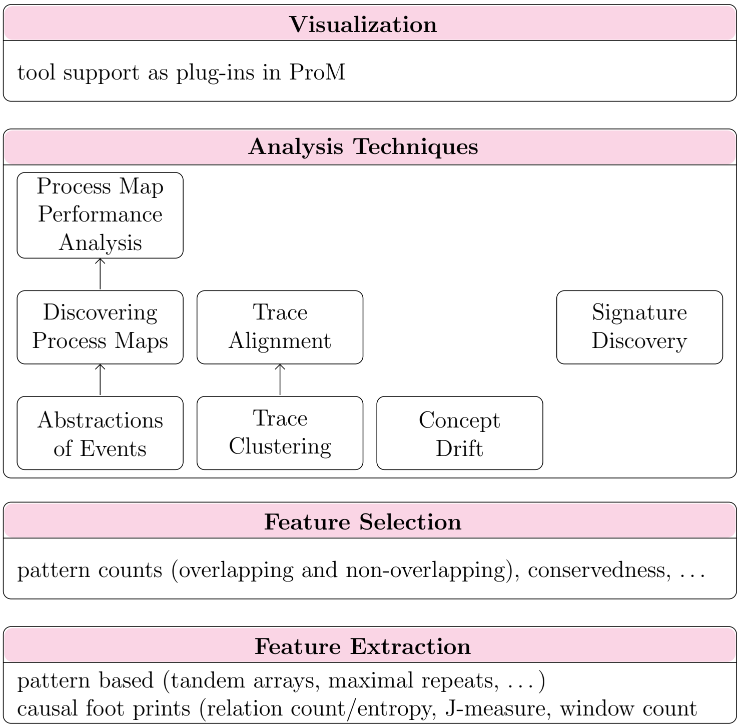

Visualization
tool support as plug-ins in ProM
Analysis Techniques
Process Map Performance Analysis
Discovering Process Maps
Trace Alignment
Signature Discovery
Abstractions of Events
Trace Clustering
Concept Drift
Feature Selection
pattern counts (overlapping and non-overlapping), conservedness, . . .
Feature Extraction
pattern based (tandem arrays, maximal repeats, . . .)
causal foot prints (relation count/entropy, J-measure, window count

Figure 12.1: Contributions of this thesis.

we presented three techniques, viz., abstractions of events, trace clustering, and concept drift, for preprocessing event logs as means of dealing with fine-granular events, heterogeneity in event logs, and process changes. Furthermore, we presented four techniques, viz., discovering process maps, process map performance analysis, trace alignment, and signature discovery, leading to advancements in process mining. All of the techniques presented in this thesis have been realized as plug-ins in ProM. These provide rich interactive visualization capabilities to explore the mined results.

Since the focus of this thesis is on enabling process mining for large scale event logs, scalability is an implicit requirement. The techniques presented in this thesis have been shown to be scalable either empirically or theoretically.

12.2 Reflection, Challenges, and Directions for Future Work

We have presented the limitations and extensions for most of the techniques proposed in this thesis in the respective chapters. In addition, we have presented an outlook for topics such as *concept drift* and *trace alignment*. In this section, we take an overarching perspective of the concepts presented in this thesis and address some challenges and interesting extensions. The reader is referred to the process mining manifesto [167] for a list of additional challenges in process mining.

From Sequences to Partial Orders

The pattern definitions (tandem arrays and maximal repeats) considered in this thesis are restricted to *sequence patterns*, i.e., continuous manifestation of entities (activities, resources, etc.) in event logs. Sequence patterns have limitations in dealing with concurrency as events involved in concurrency have an interleaved manifestation in traces. In this thesis, we proposed notions of non-continuous manifestation of patterns to capture interleavings and the grouping of patterns sharing the same alphabet into equivalence classes as a means of dealing with concurrency. However, this assumes that we uncover at least one repeat pattern where the activities involved in concurrency are not interleaved with other activities. A natural extension of sequence patterns is to consider *partial orders* or *episodes* [145] and there have been initial attempts at using episodes to mine structural workflow patterns [75–77]. However, there are still many open issues and finding/recognizing concurrent events remains a challenging problem, e.g., dealing with the sensitivity to the size of episode windows. This warrants a holistic perspective of patterns by analyzing traces not just from a control-flow perspective but also considering the data and resource perspectives. Such a holistic perspective might help in dealing with concurrency and resolving ambiguities.

Artifact-Centric Process Mining

Experiences from applying process discovery techniques on real-life logs revealed that viewing processes as a *flat single monolithic* entity is problematic. In reality, processes are designed and executed in a modular way, e.g., we have seen in the AMC case study that the patient treatment procedures are not monolithic but a collection of processes from multiple interacting departments, each with concrete and well-defined responsibilities/functionalities. This modeling paradigm, where processes are modeled as a collection of interacting entities, called as *Proclets* [226, 227] and *artifact-centric* modeling [37, 66, 161], is gaining prominence among BPM practitioners in recent years. Several new challenges emerge for process mining to cater to artifact-centric processes. For example, there does not exist a unique notion of a case or process instance as the process cannot be considered in isolation, several different cases overlap and synchronize at different points, etc. Figure 12.2 depicts a scenario where there is an ambiguity in associating the instances of activity `b` to instances of activity `a`. It is to be resolved whether the first occurrence of `b` corresponds to the first or the second occurrence of `a`.

In this thesis, we have considered abstractions as a means of defining modules and our two-phase approach to process discovery aims at viewing the process as a collection of process models (subprocesses). However, this is a restricted view of artifacts and there is a need for new process mining techniques that look at

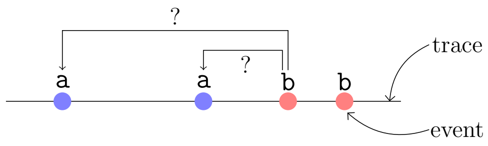


Figure 12.2: Ambiguities in associating events related to multiple instances.

artifact-centric models holistically. Recent efforts in process mining point towards this emerging area of research [1, 65, 124].

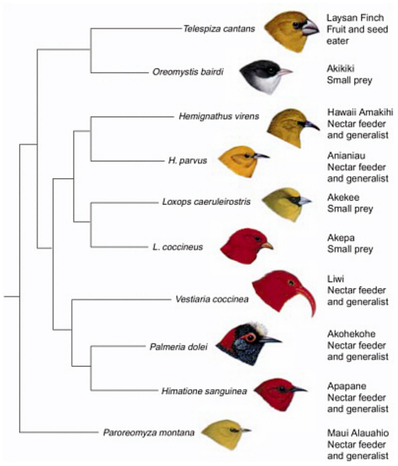
Operational Support

Process mining can be used off-line and online. The majority of research in process mining is focussed on off-line (post-mortem) analysis. Online process mining, focusing on providing *operational support* such as predictions and recommendations for running cases, is gaining attention in recent years [234, 236]. The topics of *concept drift* and *signature discovery* have a natural fit for operational support. Although detecting concept drifts is important for off-line analysis, it is more interesting and appropriate for online analysis, e.g., an organization would be more interested in knowing a change in behavior of their customers or a change in demand *as and when* it is happening. Such real-time triggers help them to take quick remedial actions and avoid any ill-effects of the change. Similarly, we have looked at signature discovery as a means of *reactive* assistance. We can extend the basic concept to enable *predictive* assistance, e.g., signs of emergence of a signature pattern can be used to predict an impending part failure or an insurance fraud. Extending techniques presented in this thesis for operational support is not straightforward and new challenges emerge. For example, for being able to detect drifts online, the number of samples required remains an issue; for signature discovery, one should be able to uncover early symptoms that ultimately manifest in an unexpected behavior. Furthermore, one needs additional computational power and efficient techniques to do such analysis in near real-time.

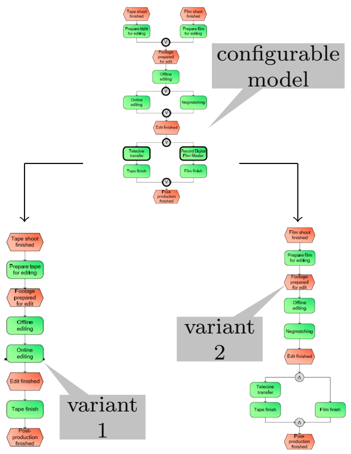
When Process Mining Meets Bioinformatics

Some of the techniques presented in this thesis (abstractions of events, trace clustering, and trace alignment) are inspired from techniques in bioinformatics (or sequence informatics, in a broader sense). Just like process mining, which aims at extracting non-trivial process related knowledge from event logs, bioinformatics aims at increasing the understanding of biological processes through the analysis of information associated with biological molecules, often represented as DNA/protein sequences [140]. It is important to note that, to a large extent, sequence analysis is a fundamental aspect in almost all facets of process mining and bioinformatics. In spite of all the peculiarities specific to business processes and process mining, the relatively young field of process mining should, in our view, take account of the conceptual foundations, practical experiences, and analysis tools developed by sequence informatics researchers over the last couple of decades.

In this thesis, we have shown some initial successes, which demonstrate that process mining techniques can benefit from such a cross-fertilization. For example, initial attempts at such a crossover enabled the discovery of hierarchical process models and helped extend the scope of conformance checking to also cover the direct inspection of traces. Furthermore, there are many unexplored areas where bioinformatics and process mining share some common goals. One such example is the commonality between phylogenetics and process configuration. Phylogenetics refers to the study of evolutionary relationships. A phylogeny is a tree representation of the evolutionary history of a set (family) of organisms, gene/protein sequences, etc. The basic premise in phylogenetics is that genes have evolved by duplication and divergence from common ancestors [211] and therefore can exist in a nested hierarchy of relatedness. Figure 12.3(a) depicts the phylogeny of some of the species of Hawaiian honeycreeper [166]. These variant species descended from a single ancestor over the last ten million years. Likewise, process configuration [233] is primarily concerned with managing families of business processes that are similar to one another in many ways yet differing in some other ways. For example, processes within different municipalities are very similar in many aspects and differ in some other aspects. A *configurable process model* describes a family of similar process models in a given domain [233], and can be thought of as the genesis (root) of the family. All variants in the family can be derived from the configurable model through a series of change patterns [261]. Figure 12.3(b) (adapted from [186]) depicts an example of a configurable model (parent) and two variants (children) derived from it. One of the core research problems in process configuration is to automatically derive configurable process models from specific models and event logs.



(a) Phylogeny of the Hawaiian honeycreeper.



(b) Process configuration

Figure 12.3: Similarity between phylogeny and process configuration.

One can find stark similarity between phylogenetics and process configuration. Techniques have been proposed in the bioinformatics literature to discover phylogenies both from (protein) structures as well as from sequences. This can be compared to deriving configurable process models from specific models and from event logs respectively. The adaptability of phylogeny construction techniques to process configuration needs to be explored. Experiences from bioinformatics can also contribute to tooling (e.g., visualization) and infrastructure efforts (e.g., benchmarking and advanced data repositories) in process mining. The reader is referred to [20, 21] for a more comprehensive discussion on the synergy between these two disciplines. Such an overlap between the goals, combined with the promising initial results, calls for a more rigorous attempt at understanding and exploiting the synergy between these two disciplines.

Learnability and Sample Complexity

The goodness of any process mining result largely depends on the characteristics of the data and the characteristics of the analysis algorithm. However, with the exception of the α-algorithm [229], none of the existing algorithms in process mining elicit their requirements/characteristics in a sufficient manner. The impact of this is felt in two ways: firstly, analysts are unaware of which techniques can provide meaningful results for the data at their disposal and secondly, algorithms do not apriori know whether they can learn a concept (be it process discovery, conformance or any other aspect of process mining) from the provided data. The effect of this is reflected in the incomprehensible and unsatisfactory results, often produced, as shown in many of the examples in this thesis. We perceive a lack of emphasis on these aspects to be one of the shortcomings in process mining research that needs to be seriously addressed.

We believe that process mining can benefit from concepts developed in computational learning theory [122, 141], a mathematical field related to the analysis of machine learning algorithms. There are two main aspects that learning theory tries to address: (i) *learnability* and (ii) *sample complexity*. The former is concerned with *learning* a "good" model of the world from some information about the "world", usually provided in the form of data drawn from some unknown underlying distribution. The latter deals with issues surrounding the necessary and sufficient number of samples needed to form a "good" model. Learning theory enables the quantification of "good" precisely depending on the specific problem at hand, e.g., the PAC (Probably Approximately Correct) learning paradigm [215] addresses the learnability of a model within a desired level of accuracy with high probability from randomly selected training examples. Both of these aspects are relevant to process mining. For example, the learnability problem in process discovery can be posed as being able to mine a process model with a desired level of fitness, precision, generalization, and simplicity given an event log with certain well-defined characteristics, while the sample complexity refers to the size of the event log (the number of traces) needed for the discovery algorithm to learn such a model.

Gold [82] initially formalized the problem of language learnability as a convergence to a grammar from an infinite stream of sequence data and showed that even regular languages cannot be exactly learned from positive examples only. The PAC learning paradigm [215] improves significantly on the Gold model and offers a more plausible treatment of convergence. Other extensions to this model that improve learnability have been proposed [36]. The focus of most formal models of learnability is on investigating which classes of languages can be learned from finite or infinite data, without being concerned about the efficiency of learning [7]. In the context of process mining, this concept was initially raised in the seminal work of Cook and Wolf [39]. However, for reasons unknown to us, it was not pursued subsequently by any researcher. This topic has started gaining attention recently [256, 262], which is a welcome sign. Looking at process mining algorithms from a learning theory perspective enables a well-founded theoretical framework upon which to reason upon, compare, and evaluate different algorithms, and comment on their learnability.

On a closing note, process mining has had a remarkable journey so far. The initial promise and successes have led to more and more organizations look up to process mining as a *panacea* for many different purposes. The contributions made in this thesis extends the journey by enabling process mining for large scale event logs. Nonetheless, there are many evolving challenges for process mining researchers to address making the journey ahead one of adventure and discovery.

Appendix A
Classification of Patient Diagnosis

Table A.1: Description of diagnosis of patients. The stage of malignancy is not known for some cases in all the diagnosis categories.

Diagnosis Value	Diagnosis Description
M11	Pertains to the cancer of the vulva. Describes information on cases diagnosed with • squamous cell carcinoma (stages I, II, III1, III2, IVa and IVb) • malignant neoplasms and melanoma • basal cell carcinoma • borderline malignancy
M12	Pertains to the cancer of the vagina. Describes information on cases diagnosed with • squamous cell carcinoma (stages II, III and IVb) • malignant neoplasms • adenocarcinoma (stage II) Certain metastases cases are also included.
M13	Pertains to the cancer of the cervix (uteri). Describes information on cases diagnosed with • squamous cell carcinoma (stages Ia1, Ia2, Ib, IIa, IIb, IIIb, IVa and IVb) • malignant neoplasms • adenocarcinoma (stages Ia1, Ib and IIa) • borderline malignancy • sarcoma

M14

Pertains to the cancer of the corpus uteri. Describes information on cases diagnosed with

- adenocarcinoma (stages Ia, Ib, Ic, IIa, IIb, IIIa, IIIb, IVa and IVb)
- malignant neoplasms and endometrium
- clear cell carcinoma (stages Ib and IIIb)
- borderline malignancy

Certain metastases cases are also included.

M15

Primarily pertains to the cancer of the corpus uteri of type *sarcoma* (stages II and III according to the FIGO staging system). However, certain cases of colon cancer and myometrium are also classified into this category

M16

Pertains to the cancer of the ovary. Describes information on cases diagnosed with

- adenocarcinoma of types
  - serous (stages Ia, Ic, IIa, IIIb, IIIc and IV)
  - endometroid (stages Ic, IIIc)
  - mucinous (stages Ic, IIc and IIIc)
  - non-differentiated (stages IIIc and IV)
- non-epithelial malignancy (stages Ia, IIa, IIIa and IIIc)
- neoplasms
- borderline malignancy
- clear cell carcinoma.

Certain metastases cases are also included.

821

Pertains to the cancer of the ovary. Describes information on cases diagnosed with

- adenocarcinoma of types
  - serous (stage IIIc)
  - mucinous (stage IIIc)
- non-epithelial malignancy
- neoplasms

822	Pertains to the cancer of the cervix (uteri). Describes information on cases diagnosed with • squamous cell carcinoma (stage Ib) • adenocarcinoma (stages IIa and IIb) • borderline malignancy • malignant neoplasms
106	Describes a heterogeneous mix of cases pertaining to the cancers of • cervix uteri, of types squamous cell carcinoma (stages Ia and IIa), malignant neoplasms and borderline malignancy • vulva, of types squamous cell carcinoma (stages III2, IVa and IVb) and malignant melanoma • corpus uteri, of types adenocarcinoma (stages Ib, Ic and IIa), malignant neoplasms and borderline malignancy • vagina, endometrium and ovarian tube
823	Describes a heterogeneous mix of cases pertaining to the cancers of • corpus uteri, of types adenocarcinoma (stages IVa and IVb), malignant neoplasms and sarcoma (stage IVb according to the FIGO staging system) • ovary, of type serous adenocarcinoma (stage IIIc) • endometrium
839	Describes a heterogeneous mix of cases pertaining to the cancers of • ovary, of types serous adenocarcinoma (stages IIIc and IV) and borderline malignany • uterine appendages, of type malignant neoplasms • vulva, of type malignant neoplasms

Table A.2: Distribution of cases based on the values for diagnosis code combination. Each case in the event log may contain up to 16 diagnosis code attributes. The different diagnosis code attributes can take on values such as M11, M13 and 106. This table depicts the values of diagnosis code combinations manifested in the cases in the event log and the number of cases for each combination.

Diagnosis Value Combination	Number of Cases
{M13}	252
{M16}	201
{M11}	162
{M14}	106
{106}	70
{822, 106}	61
{M13, 106}	57
{M13, 822, 106}	47
{106, M14}	31
{M16, 821}	25
{M12}	17
{821}	16
{M15}	14
{822}	11
{M11, 106}	11
{M13, 822}	10
{839}	8
{823}	7
{839, M16}	6
{M12, 106}	4
{823, 106, M14}	4
{823, M14}	3
{106, 821}	3
{M11, 106, 839}	2
{106, M15}	2
{106, 839}	1
{M13, 106, 839}	1
{M12, 839}	1
{822, 106, M14}	1
{M13, 106, 821}	1
{839, M16, 821}	1
{823, 106}	1
{106, M16, 821}	1
{823, 106, M15}	1
{106, M16}	1
{822, 106, 821}	1
{839, 821}	1
{M11, 822}	1

Appendix B Publications

1. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Context Aware Trace Clustering: Towards Improving Process Mining Results. In H. Liu and Z. Obradovic, editors, *Proceedings of the SIAM International Conference on Data Mining (SDM 2009)*, pages 401–412. Society for Industrial and Applied Mathematics, 2009.
2. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Abstractions in Process Mining: A Taxonomy of Patterns. In U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors, *Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 159–175. Springer-Verlag, Berlin, 2009.
3. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models. In S. Rinderle-Ma, S. Sadiq, and F. Leymann, editors, *BPM 2009 Workshops, Proceedings of the Fifth Workshop on Business Process Intelligence (BPI 2009)*, volume 43 of *Lecture Notes in Business Information Processing*, pages 170–181. Springer-Verlag, Berlin, 2010.
4. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Trace Alignment in Process Mining: Opportunities for Process Diagnostics. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*, pages 227–242. Springer-Verlag, Berlin, 2010. [**Awarded as the Best Student Paper**]
5. J. Li, R.P. Jagadeesh Chandra Bose, and W.M.P. van der Aalst. Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI 2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 109–121. Springer-Verlag, Berlin, 2011.
6. R.P. Jagadeesh Chandra Bose, W.M.P. van der Aalst, I. Žliobaitė, and M. Pechenizkiy. Handling Concept Drift in Process Mining. In H. Mouratidis and C. Rolland, editors, *International Conference on Advanced Information Systems Engineering (CAiSE 2011)*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, Berlin, 2011.
7. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. When Process Mining Meets Bioinformatics. In S. Nurcan, editor, *Proceedings of the CAiSE Forum*

2011, volume 734 of *CEUR Workshop Proceedings*, pages 147–154. CEUR-WS.org, 2011.

8. R.P. Jagadeesh Chandra Bose, E.H.M.W. Verbeek, and W.M.P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In S. Nurcan, editor, *Proceedings of the CAiSE Forum 2011*, volume 734 of *CEUR Workshop Proceedings*, pages 33–40. CEUR-WS.org, 2011.
9. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Analysis of Patient Treatment Procedures. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *BPM 2011 Workshops, Proceedings of the Seventh Workshop on Business Process Intelligence (BPI 2011)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 165–166. Springer-Verlag, Berlin, 2012. [**Winner of the First Business Process Intelligence Challenge**]
10. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges. *Information Systems*, 37(2):117–141, 2012.
11. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. When Process Mining Meets Bioinformatics. In S. Nurcan, editor, *Proceedings of the CAiSE Forum 2011*, volume 107 of *Lecture Notes in Business Information Processing*, pages 202–217. Springer-Verlag, Berlin, 2012.
12. R.P. Jagadeesh Chandra Bose, E.H.M.W. Verbeek, and W.M.P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In S. Nurcan, editor, *Proceedings of the CAiSE Forum 2011*, volume 107 of *Lecture Notes in Business Information Processing*, pages 33–48. Springer-Verlag, Berlin, 2012.

Bibliography

[1] ACSI. Artifact-Centric Service Interoperation (ACSI). Project Home Page. `www.acsi-project.eu`. (cited on p. 323)

[2] A. Adriansyah. Performance Analysis of Business Processes from Event Logs and Given Process Models. Master's thesis, Eindhoven University of Technology, 2009. (cited on pp. 174, 175, and 178)

[3] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 55–64, 2011. (cited on pp. 11 and 200)

[4] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *Business Process Mangement Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011. (cited on pp. 11 and 200)

[5] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, volume 1215, pages 487–499, 1994. (cited on pp. 229, 238, and 239)

[6] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Work flow Logs. In I. Ramos, G. Alonso, H. J. Schek, and F. Saltor, editors, *Advances in Database Technology–EDBT 98*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998. (cited on pp. 139 and 140)

[7] A. Alishahi. Computational Modeling of Human Language Acquisition. In G. Hirst, editor, *Synthesis Lectures on Human Language Technologies*, volume 3, pages 1–107. Morgan & Claypool Publishers, 2010. (cited on p. 326)

[8] S. F. Altschul. Amino Acid Substitution Matrices from an Information Theoretic Perpsective. *Journal of Molecular Biology*, 219(3):555–565, 1991. (cited on p. 87)

[9] K. Anyanwu, A. P. Sheth, J. Cardoso, J. A. Miller, and K. Kochut. Healthcare Enterprise Process Development and Integration. *Journal of Research and Practice in Information Technology*, 35(2):83–98, 2003. (cited on p. 267)

[10] D. J. Bacon and W. F. Anderson. Multiple Sequence Alignment. *Journal of Molecular Biology*, 191(2):153–161, 1986. (cited on p. 206)

[11] E. Badouel and P. Darondeau. Theory of Regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer-Verlag, Berlin, 1998. (cited on p. 141)

[12] J. L. Benedet, H.Bender, H. Jones, H. Y. Ngan, and S. Pecorelli. FIGO Staging Classification and Clinical Practice Guidelines in the Management of Gynecologic Cancers.

FIGO Committee on Gynecologic Oncology. *International Journal of Gynaecology and Obstetrics*, 70(2):209–262, 2000. (cited on p. 269)

[13] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007. (cited on p. 141)

[14] A. Bertie. Java Statistical Classes. `http://www.jsc.nildram.co.uk`. (cited on p. 253)

[15] F. Bezerra and J. Wainer. Anomaly Detection Algorithms in Logs of Process Aware Systems. In R. L. Wainwright and H. Haddad, editors, *Proceedings of the ACM Symposium on Applied Computing (SAC 2008)*, pages 951–952. ACM, 2008. (cited on p. 11)

[16] A. Bifet and R. Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining (SDM)*, pages 443–448, 2007. (cited on pp. 114 and 123)

[17] E. Bingham and H. Mannila. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 245–250. ACM, 2001. (cited on p. 123)

[18] N. M. Blachman. The Amount of Information that y Gives About X. *IEEE Transactions on Information Theory*, IT-14(1):27–31, 1968. (cited on p. 120)

[19] R. J. Bolton and D. J. Hand. Statistical Fraud Detection: A Review. *Statistical Science*, 17(3):235–249, 2002. (cited on p. 228)

[20] R. P. J. C. Bose and W. M. P. van der Aalst. When Process Mining Meets Bioinformatics. In S. Nurcan, editor, *Proceedings of the CAiSE Forum*, volume 734 of *CEUR Workshop Proceedings*, pages 147–154. CEUR-WS.org, 2011. (cited on p. 325)

[21] R. P. J. C. Bose and W. M. P. van der Aalst. When Process Mining Meets Bioinformatics. In S. Nurcan, editor, *CAiSE Forum 2011*, volume 107 of *Lecture Notes in Business Information Processing*, pages 202–217. Springer-Verlag, Berlin, 2012. (cited on p. 325)

[22] R. P. J. C. Bose, W. M. P. van der Aalst, I. Žliobaitė, and M. Pechenizkiy. Handling Concept Drift in Process Mining. In H. Mouratidis and C. Rolland, editors, *International Conference on Advanced Information Systems Engineering (CAiSE 2011)*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, Berlin, 2011. (cited on p. 10)

[23] R. P. J. C. Bose, E. H. M. W. Verbeek, and W. M. P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In S. Nurcan, editor, *Proceedings of the CAiSE Forum*, volume 734, pages 33–40. CEUR-WS.org, 2011. (cited on p. 255)

[24] R. P. J. C. Bose, E. H. M. W. Verbeek, and W. M. P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In S. Nurcan, editor, *CAiSE Forum 2011*, volume 107 of *Lecture Notes in Business Information Processing*, pages 33–48. Springer-Verlag, Berlin, 2012. (cited on p. 255)

[25] A. T. Bouloutas, S. Calo, and A. Finkel. Alarm Correlation and Fault Identification in Communication Networks. *IEEE Transactions on Communications*, 42(234):523–533, 1994. (cited on p. 229)

[26] M. Bozkaya, J. Gabriels, and J. M. van der Werf. Process Diagnostics: A Method Based on Process Mining. In A. Kusiak and S. Lee, editors, *Proceedings of the International Conference on Information, Process, and Knowledge Management (eKNOW 2009)*, pages 22–27. IEEE Computer Society, 2009. (cited on p. 11)

[27] N. Brand and H. Van der Kolk. *Workflow Analysis and Design*. Kluwer Bedrijfswetenschappen, Deventer, 1995. (cited on p. 173)

[28] J. C. A. M. Buijs, B. F van Dongen, and W. M. P. van der Aalst. Towards Cross-Organizational Process Mining in Collections of Process Models and thier Executions. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 2–13. Springer-Verlag, Berlin, 2012. (cited on p. 284)

[29] C. Pedrinaci and J. Domingue. Towards an Ontology for Process Monitoring and Mining. In M.Hepp, K. Hinkelmann, D. Karagiannis, R. Klein, and N. Stojanovic, editors, *Semantic Business Process and Product Lifecycle Management*, volume 251, pages 76–87. CEUR-WS.org, 2007. (cited on p. 51)

[30] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Model-Based Clustering and Visualization of Navigation Patterns on a Web Site. *Data Mining and Knowledge Discovery*, 7(4):399–424, 2003. (cited on p. 80)

[31] J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In M. Dumas, R. Manfred, and M. C. Shan, editors, *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 358–373. Springer-Verlag, Berlin, 2008. (cited on p. 141)

[32] H. Carrillo and D. Lipman. The Multiple Sequence Alignment Problem in Biology. *SIAM Journal of Applied Mathematics*, 48(5):1073–1082, 1988. (cited on p. 206)

[33] C. C. Chang and C. J. Lin. LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. (cited on p. 260)

[34] H. Chen, G. Jiang, C. Ungureanu, and K. Yoshihira. Online Tracking of Component Interactions for Failure Detection and Localization in Distributed Systems. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, 37(4):644–651, 2007. (cited on p. 229)

[35] C. F. Cheung, J. X. Yu, and H. Lu. Constructing Suffix Tree for Gigabyte Sequences with Megabyte Memory. *IEEE Transactions on Knowledge and Data Engineering*, 17 (1):90–105, 2005. (cited on p. 56)

[36] A. Clark and S. Lappin. Unsupervised Learning and Grammar Induction. In S. Lappin A. Clark, C. Fox, editor, *The Handbook of Computational Linguistics and Natural Language Processing*, pages 197–220. Blackwell Publishing Ltd., 2010. (cited on p. 326)

[37] D. Cohn and R. Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009. (cited on p. 322)

[38] J. E. Cook. *Process Discovery and Validation Through Event-Data Analysis*. PhD thesis, University of Colorado, 1996. (cited on p. 139)

[39] J. E. Cook and A. L. Wolf. Automating Process Discovery Through Event-Data Analysis. In *Proceedings of the 17th International Conference on Software Engineering (ICSE)*, pages 73–82. ACM Press, 1995. (cited on pp. 139 and 326)

[40] J. E. Cook and A. L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE)*, pages 35–45. ACM Press, 1998. (cited on p. 139)

[41] J. E. Cook and A. L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3): 215–249, 1998. (cited on p. 139)

[42] J. E. Cook, Z. Du, C. Liu, and A. L. Wolf. Discovering Models of Behavior for Concurrent Workfows. *Computers in Industry*, 53(3):297–319, 2004. (cited on p. 139)

[43] G. Cormode, S. Muthukrishnan, and S. C. Sahinalp. Permutation Editing and Matching via Embeddings. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 481–492. Springer-Verlag, Berlin, 2001. (cited on p. 152)

[44] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998. (cited on p. 51)

[45] CoSeLog. Configurable Services for Local Governments. Project Home Page. `www.win.tue.nl/coselog`. (cited on p. 284)

[46] T. Cover and P. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. (cited on pp. 236 and 260)

[47] C. Daniel, D. Paul, M. Vidhya, O. Marco, H. Eun-Jong, W. Yaoyu, S. Shyamal, C. Brian, P. Shobha, and H. Enoch. PFAAT version 2.0: A Tool for Editing, Annotating, and Analyzing Multiple Sequence Alignments. *BMC Bioinformatics*, 8(1):381, 2007. (cited on p. 201)

[48] M. K. Das and H. K. Dai. A Survey of DNA Motif Finding Algorithms. *BMC Bioinformatics*, 8(Suppl 7):S21, 2007. (cited on p. 47)

[49] A. K. A. de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, 2006. (cited on pp. 97, 101, 103, 140, and 159)

[50] A. K. A. de Medeiros and W. M. P. van der Aalst. Process Mining Towards Semantics. In T. S. Dillon, E. Chang, R. Meersman, and K. Sycara, editors, *Advances in Web Semantics-I*, volume 4891 of *Lecture Notes in Computer Science*, pages 35–80. Springer-Verlag, Berlin, 2009. (cited on p. 224)

[51] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process Mining: Extending the α-algorithm to Mine Short Loops. Technical Report BETA Working Paper Series, WP 113, Eindhoven University of Technology, 2004. (cited on pp. 140 and 155)

[52] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm. In L. Baresi, S. Dustdar, H. C. Gall, and M. Matera, editors, *Ubiquitous Mobile Information and Collaboration Systems*, volume 3272 of *Lecture Notes in Computer Science*, pages 151–165. Springer-Verlag, Berlin, 2005. (cited on p. 140)

[53] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007. (cited on pp. 97, 101, 103, 140, and 159)

[54] A. K. A. de Medeiros, A. Guzzo, G. Greco, W. M. P. van der Aalst, A. J. M. M. Weijters, B. F. van Dongen, and D. Sacca. Process Mining Based on Clustering: A Quest for Precision. In A. H. M. ter Hofstede, B. Benatallah, and H. Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, Berlin, 2008. (cited on pp. 12 and 79)

[55] A. K. A. de Medeiros, W. M. P. van der Aalst, and P. Carlos. Semantic Process Mining Tools: Core Building Blocks. In W. Golden, T. Acton, K. Conboy, H. van der Heijden, and V. K. Tuunainen, editors, *Proceedings of the* 16^{th} *European Conference on Information Systems (ECIS 2008)*, pages 1953–1964, 2008. (cited on pp. 34, 51, and 224)

[56] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977. (cited on p. 80)

[57] R. A. Derrig. Insurance Fraud. *The Journal of Risk and Insurance*, 69(3):271–287, 2002. (cited on p. 228)

[58] J. Desel and W. Reisig. The synthesis problem of petri nets. *Acta Informatica*, 33(4): 297–315, 1996. (cited on p. 51)

[59] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic Co-clustering. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 89–98. ACM, 2003. (cited on p. 79)

[60] J. C. Dunn. Well Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics*, 4(1):95–104, 1974. (cited on p. 96)

[61] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 2002. (cited on p. 206)

[62] R. C. Edgar. MUSCLE: A Multiple Sequence Alignment with Reduced Time and Space Complexity. *BMC Bioinformatics*, 5:113, 2004. (cited on pp. 95, 218, and 250)

[63] R. C. Edgar and S. Batzoglou. Multiple Sequence Alignment. *Current Opinion in Structural Biology*, 16(3):368–373, 2006. (cited on p. 206)

[64] R. Elwell and R. Polikar. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011. (cited on p. 114)

[65] D. Fahland, M. De Leoni, B. F. van Dongen, and W. M. P. van der Aalst. Behavioral Conformance of Artifact-Centric Process Models. In W. Abramowicz, editor, *Business Information Systems*, volume 87 of *Lecture Notes in Business Information Processing*, pages 37–49. Springer-Verlag, Berlin, 2011. (cited on p. 323)

[66] D. Fahland, M. De Leoni, B. F. van Dongen, and W. M. P. van der Aalst. Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS (2011)*, volume 705 of *CEUR Workshop Proceedings*, pages 9–15. CEUR-WS.org, 2011. (cited on p. 322)

[67] D. F. Feng and R. F. Doolittle. Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees. *Journal of Molecular Evolution*, 25(4):351–360, 1987. (cited on pp. 201 and 206)

[68] D. F. Feng and R. F. Doolittle. Progressive Alignment of Amino Acid Sequences and Construction of Phylogenetic Trees from Them. *Methods in Enzymology*, 266:368–382, 1996. (cited on pp. 201 and 206)

[69] D. R. Ferreira. Applied Sequence Clustering Techniques for Process Mining. In J. Cardoso and W. M. P. van der Aalst, editors, *Handbook of Research on Business Process Modeling*, pages 492–513. ISI Global, 2009. (cited on pp. 80 and 105)

[70] D. R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira. Approaching Process Mining with Sequence Clustering: Experiments and Findings. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 360–374. Springer-Verlag, Berlin, 2007. (cited on pp. 80, 98, 99, and 105)

[71] I. K. Fodor. A Survey of Dimensionality Reduction Techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, pages 1–24, 2002. (cited on pp. 123 and 132)

[72] F. Folino, G. Greco, A. Guzzo, and L. Pontieri. Mining Usage Scenarios in Business Processes: Outlier-Aware Discovery and Run-Time Prediction. *Data & Knowledge Engineering*, 70(12):1005–1029, 2011. (cited on pp. 79, 101, 209, 224, 229, and 230)

[73] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall New York, 1982. (cited on pp. 83 and 86)

[74] K. S. Fu and S. Y. Lu. A Clustering Procedure for Syntactic Patterns. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(10):734–742, 1977. (cited on pp. 83 and 86)

[75] W. Gaaloul and C. Godart. Mining Workflow Recovery from Event Based Logs. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 169–185. Springer-Verlag, Berlin, 2005. (cited on p. 322)

[76] W. Gaaloul, S. Bhiri, and C. Godart. Discovering Workflow Transactional Behavior from Event-Based Log. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 3–18. Springer-Verlag, Berlin, 2004. (cited on p. 322)

[77] W. Gaaloul, S. Alaoui, K. Baina, and C. Godart. Mining Workflow Patterns through Event-Data Analysis. In *IEEE/IPSJ Symposium on Applications and the Internet Workshops, Saint 2005 Workshops*, pages 226–229. IEEE Computer Soceity, 2005. (cited on p. 322)

[78] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In *In SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer-Verlag, Berlin, 2004. (cited on p. 114)

[79] J. Gantz and D. Reinsel. The Digital Universe Decade–Are You Ready? White Paper, IDC, Sponsored by EMC Corporation, May, 2010. (cited on p. 3)

[80] L. Ghionna, G. Greco, A. Guzzo, and L. Pontieri. Outlier Detection Techniques for Process Mining Applications. In A. An, S. Matwin, Z. W. Ras, and D. Slezak, editors, *Foundations of Intelligent Systems*, volume 4994 of *Lecture Notes in Computer Science*, pages 150–159. Springer-Verlag, Berlin, 2008. (cited on pp. 79, 101, 209, and 224)

[81] M. Golani and S. S. Pinter. Generating a Process Model from a Process Audit Log. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 136–151. Springer-Verlag, Berlin, 2003. (cited on p. 140)

[82] E. M. Gold. Language Identification in the Limit. *Information and Control*, 10(5): 447–474, 1967. (cited on p. 326)

[83] C. Görg, M. Pohl, E. Qeli, and K. Xu. Visual Representations. In A. Kerren, A. Ebert, and J. Meye, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes in Computer Science*, pages 163–230. Springer-Verlag, Berlin, 2007. (cited on pp. 10 and 137)

[84] O. Gotoh. Multiple Sequence Alignment: Algorithms and Applications. *Advances in Biophysics*, 36:159–206, 1999. (cited on p. 201)

[85] C. Grasso and C. Lee. Combining Partial Order Alignment and Progressive Multiple Sequence Alignment Increases Alignment Speed and Scalability to Very Large Alignment Problems. *Bioinformatics*, 20(10):1546–1556, 2004. (cited on p. 225)

[86] G. Greco, A. Guzzo, and L. Pontieri. Mining Hierarchies of Models: From Abstract Views to Concrete Specifications. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, Berlin, 2005. (cited on pp. 12 and 142)

[87] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006. (cited on pp. 12, 79, and 142)

[88] G. Greco, A. Guzzo, and L. Pontieri. An Information Theoretic Framework for Process Structure and Data Mining. *International Journal of Data Warehousing and Mining*, 3(4):99–119, 2007. (cited on pp. 79 and 107)

[89] G. Greco, A. Guzzo, and L. Pontieri. Mining Taxonomies of Process Models. *Data & Knowledge Engineering*, 67(1):74–102, 2008. (cited on pp. 12 and 142)

[90] V. Grover, W. J. Kettinger, and J. T. C. Teng. Business Process Change in the 21st Century. *Business & Economic Review*, 46(2):14–18, 2000. (cited on p. 109)

[91] C. W. Günther. *Process Mining in Flexible Environments.* PhD thesis, Eindhoven University of Technology, 2009. (cited on pp. 34, 42, 43, 44, 174, and 200)

[92] C. W. Günther. XES Standard Definition, 2009. `www.xes-standard.org`. (cited on pp. 33, 36, 37, and 247)

[93] C. W. Günther and W. M. P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, Berlin, 2006. (cited on p. 34)

[94] C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007. (cited on pp. 9, 15, 43, 52, 137, 142, 160, 165, 174, 255, and 307)

[95] C. W. Günther, S. Rinderle-Ma, M. Reichert, and W. M. P. van der Aalst. Using Process Mining to Learn from Process Changes in Evolutionary Systems. *International Journal of Business Process Integration and Management*, 3(1):61–78, 2008. (cited on pp. 14, 109, and 114)

[96] C. W. Günther, A. Rozinat, and W. M. P. van der Aalst. Activity Mining by Global Trace Segmentation. In S. Rinderle-Ma, S. Sadiq, and Frank Leymann, editors, *Business Process Mangement Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 128–139. Springer-Verlag, Berlin, 2010. (cited on p. 51)

[97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997. (cited on pp. 11, 47, 57, 60, and 152)

[98] D. Gusfield and J. Stoye. Linear Time Algorithms for Finding and Representing all the Tandem Repeats in a String. *Journal of Computer and System Sciences*, 69(4): 525–546, 2004. (cited on pp. 11, 47, 52, and 57)

[99] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003. (cited on pp. 123 and 132)

[100] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster Validity Methods: Part I. *ACM SIGMOD Record*, 31(2):40–45, 2002. (cited on p. 97)

[101] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part II. *ACM SIGMOD Record*, 31(3):19–27, 2002. (cited on p. 97)

[102] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. (cited on p. 260)

[103] M. Hammer. *Beyond Reengineering: How the Process-Centered Organization is Changing Our Work and Our Lives.* New York: Harperbusiness, 1996. (cited on p. 117)

[104] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2006. (cited on pp. 78, 79, 93, 97, 238, 239, 240, and 261)

[105] S. Henikoff and J. G. Henikoff. Amino Acid Substitution Matrices from Protein Blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89 (22):10915–10919, 1992. (cited on p. 87)

[106] J. Herbst. A Machine Learning Approach to Workflow Management. In R. L. de Mántaras and E. Plaza, editors, *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000. (cited on p. 140)

[107] J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9(2):67–92, 2000. (cited on p. 140)

[108] J. Herbst and D. Karagiannis. Workflow Mining with InWoLvE. *Computers in Industry*, 53(3):245–264, 2004. (cited on p. 140)

[109] Hewlett-Packard. HP OpenView Self-Healing Services, 2006. `www.managementsoftware.hp.com/services/selfhealing_whitepaper.pdf`. (cited on p. 229)

[110] P. T. G. Hornix. Performance Analysis of Business Processes through Process Mining. Master's thesis, Eindhoven University of Technology, 2007. (cited on pp. 174, 175, and 201)

[111] C. Huang, I. Cohen, J. Symons, and T. Abdelzaher. Achieving Scalable Automated Diagnosis of Distributed Systems Performance Problems. Technical Report HP-2006-160(R.1), Hewlett-Packard Development Company, 2007. (cited on p. 229)

[112] IBM. IBM-Integrated Service Management Software-Trivoli. `www.ibm.com/software/trivoli`. (cited on p. 229)

[113] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. (cited on pp. 52, 79, 93, and 206)

[114] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999. (cited on p. 93)

[115] M. Jans, N. Lybaert, and K. Vanhoof. Internal Fraud Risk Reduction: Results of a Data Mining Case Study. *International Journal of Accounting Information Systems*, 11(1):17–41, 2010. (cited on p. 228)

[116] M. H. Jansen-Vullers, M. W. N. C. Loosschilder, P. A. M. Kleingeld, and H. A. Reijers. Performance Measures to Evaluate the Impact of Best Practices. In B. Pernici and J. A. Gulla, editors, *Proceedings of Workshops and Doctoral Consortium of the*

19th International Conference on Advanced Information Systems Engineering (BP-MDS workshop), volume 1, pages 359–368. Tapir Academic Press Trondheim, 2007. (cited on p. 175)

[117] K. Jensen and L. M. Kristensen. *Colored Petri Nets*. Springer-Verlag, Berlin Heidelberg, 2009. (cited on pp. 42 and 43)

[118] K. Jensen, L. M. Kristensen, and L. Wells. Colored Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9(3–4):213–254, 2007. (cited on p. 42)

[119] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, Berlin, second edition, 2002. (cited on pp. 123 and 238)

[120] G. K. Kanji. *100 Statistical Tests*. SAGE Publications Ltd, 2006. (cited on p. 122)

[121] S. Karlin and S. F. Altschul. Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes. *Proceedings of the National Academy of Sciences of the United States of America*, 87(6):2264–2268, 1990. (cited on p. 87)

[122] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994. (cited on p. 325)

[123] M. Kennerly, A. Neely, and C. Adams. Survival of the Fittest: Measuring Performance in a Changing Business Enviornment. *Measuring Business Excellence*, 7(4):37–43, 2003. (cited on p. 109)

[124] R. Kikas. Discovering Mapping Between Artifact-Centric Business Process Models and Execution Logs. Master's thesis, University of Tartu, 2011. (cited on p. 323)

[125] E. Kindler, V. Rubin, and W. Schäfer. Process Mining and Petri Net Synthesis. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 105–116. Springer-Verlag, Berlin, 2006. (cited on p. 141)

[126] D. King, B. Martin, J. Dwyer, J. Healy, K. Owada, L. Smith, L. Sun, A. Van Deth, J. Wainer, and E. Willis. Review of the Gynaecological Cancers Workforce, Report for Cancer Australia. Research Report, National Institute of Labor Studies, Flinders University, Adelaide, Australia, 2008. (cited on p. 267)

[127] E. Kirkos, C. Spathis, and Y.Manolopoulos. Data Mining Techniques for the Detection of Fraudulent Financial Statements. *Expert Systems with Applications*, 32(4):995–1003, 2007. (cited on p. 228)

[128] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 1137–1145. Morgan Kaufmann, 1995. (cited on p. 240)

[129] T. Kohonen. *Self-Organizing Maps*, volume 30. Springer-Verlag, Berlin, 1995. (cited on p. 93)

[130] R. Kolpakov and G. Kucherov. Finding Maximal Repetitions in a Word in Linear Time. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 596–604, 1999. (cited on pp. 11, 47, and 57)

[131] R. Kolpakov, G. Bana, and G. Kucherov. mReps: Efficient and Flexible Detection of Tandem Repeats in DNA. *Nucleic Acids Research*, 31(13):3672–3678, 2003. (cited on pp. 11 and 47)

[132] J. Z. Kolter and M. A. Maloof. Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007. (cited on p. 114)

[133] G. N. Lance and W. T. Williams. A General Theory of Classificatory Sorting Strategies. *The Computer Journal*, 9(4):373–380, 1967. (cited on p. 96)

[134] C. Lee, C. Grasso, and M. F. Sharlow. Multiple Sequence Alignment Using Partial Order Graphs. *Bioinformatics*, 18(3):452–464, 2002. (cited on p. 225)

[135] I. Lee and R. K. Iyer. Diagnosing Rediscovered Software Problems Using Symptoms. *IEEE Transactions on Software Engineering*, 26(2):113–127, 2000. (cited on p. 229)

[136] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. (cited on pp. 75 and 83)

[137] J. Li, R. P. J. C. Bose, and W. M. P. van der Aalst. Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns. In M. zur Muehlen and J. Su, editors, *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 109–121. Springer-Verlag, Berlin, 2011. (cited on p. 255)

[138] B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In *Fourth International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 80–86. The AAAI Press, 1998. (cited on pp. 229, 238, and 239)

[139] F. Liu. Fault Diagnosis Using Process Mining. Master's thesis, Eindhoven University of Technology, 2010. (cited on p. 230)

[140] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is Bioinformatics? A Proposed Definition and Overview of the Field. *Methods of Information in Medicine*, 40 (4):346–358, 2001. (cited on p. 323)

[141] M. Anthony M. and N. Biggs. *Computational Learning Theory: An Introduction.* Cambridge University Prrress, 1997. (cited on p. 325)

[142] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer-Verlag, Berlin, 2011. (cited on pp. 18 and 201)

[143] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 192–199, 2011. (cited on p. 201)

[144] P. C. Mahalanobis. On the Generalised Distance in Statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936. (cited on p. 95)

[145] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997. (cited on pp. 52 and 322)

[146] R. S. Mans. *Workflow Support for the Healthcare Domain.* PhD thesis, Eindhoven University of Technology, 2011. (cited on p. 267)

[147] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big Data: The Next Frontier for Innovation, Competition, and Productivity. Technical report, McKinsey Global Institute, 2011. (cited on pp. 4, 9, and 319)

[148] L. Mariani and F. Pastore. Automated Identification of Failure Causes. In *Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE)*, pages 117–126. IEEE Computer Society, 2008. (cited on p. 229)

[149] A. Marzal and E. Vidal. Computation of Normalized Edit Distance and Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993. (cited on p. 92)

[150] W. J. Masek and M. S. Paterson. A Faster Algorithm Computing String Edit Distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. (cited on p. 86)

[151] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):859–874, 2011. (cited on p. 114)

[152] J. Melcher, J. Mendling, H. A. Reijers, and D. Seese. On Measuring the Understandability of Process Models. In S. Rinderle-Ma, S. W. Sadiq, and F. Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 465–476. Springer-Verlag, Berlin, 2010. (cited on p. 98)

[153] J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2009. (cited on p. 98)

[154] J. Mendling and M. Strembeck. Influence Factors of Understanding Business Process Models. In W. Abramowicz and D. Fensel, editors, *Business Information Systems*, volume 7 of *Lecture Notes in Business Information Processing*, pages 142–153. Springer-Verlag, Berlin, 2008. (cited on p. 98)

[155] B. Morgenstern, S. J. Prohaska, D. Pöhler, and P. F. Stadler. Multiple Sequence Alignment with User-defined Anchor Points. *Algorithms for Molecular Biology*, 1:6, 2006. (cited on p. 225)

[156] N. Mulyar. *Patterns for Process-Aware Information Systems: An Approach Based on Colored Petri Nets*. PhD thesis, Eindhoven University of Technology, 2009. (cited on pp. 113 and 115)

[157] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. (cited on p. 39)

[158] F. Murtagh. A Survey of Recent Advances in Hierarchical Clustering Algorithms. *The Computer Journal*, 26(4):354–359, 1983. (cited on p. 96)

[159] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. (cited on p. 204)

[160] A. Neely, M. Gregory, and K. Platts. Performance Measurement System Design: A Literature Review and Research Agenda. *International Journal of Operations & Production Management*, 25(12):1228–1263, 2005. (cited on p. 173)

[161] A. Nigam and N. S. Caswell. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 42(3):428–445, 2003. (cited on pp. 144, 273, and 322)

[162] K. Nishida and K. Yamauchi. Detecting Concept Drift using Statistical Testing. In *Proceedings of the 10th International Conference on Discovery Science (DS'07)*, pages 264–269. Springer-Verlag, Berlin, 2007. (cited on p. 114)

[163] C. Notredame. Recent Progress in Multiple Sequence Alignment: A Survey. *Pharmacogenomics*, 3(1):131–144, 2002. (cited on pp. 201 and 206)

[164] A. Oliner and J. Stearley. What Supercomputers Say: A Study of Five System Logs. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 575–584, 2007. (cited on p. 228)

[165] C. W. Olofson. Managing Data Growth Through Intelligent Partitioning: Focus on Better Database Manageability and Operational Efficiency with Sybase ASE. White Paper, IDC, Sponsored by Sybase, an SAP Company, November, 2010. (cited on p. 4)

[166] S. Olson. *Evolution in Hawaii: A Supplement to Teaching About Evolution and the Nature of Science.* National Academic Press, 2004. (cited on p. 324)

[167] IEEE Task Force on Process Mining. Process Mining Manifesto. In F. Daniel, S. Dustdar, and K. Barkaoui, editors, *BPM 2011 Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer-Verlag, Berlin, 2011. (cited on pp. 9, 15, 159, and 322)

[168] M. Pechenizkiy, J. Bakker, I. Žliobaitė, A. Ivannikov, and T. Kärkkäinen. Online Mass Flow Prediction in CFB Boilers with Explicit Detection of Sudden Concept Drift. *SIGKDD Explorations*, 11(2):109–116, 2009. (cited on p. 114)

[169] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Controls to Users.* PhD thesis, Eindhoven University of Technology, 2008. (cited on p. 201)

[170] M. Pesic and W. M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer-Verlag, Berlin, 2006. (cited on p. 201)

[171] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 287–300. IEEE Computer Society, 2007. (cited on p. 201)

[172] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962. (cited on p. 39)

[173] S. S. Pinter and M. Golani. Discovering Workflow Models from Activities Lifespans. *Computers in Industry*, 53(3):283–296, 2004. (cited on p. 140)

[174] K. Ploesser, J. C. Recker, and M. Rosemann. Towards a Classification and Lifecycle of Business Process Change. In S. Nurcan, R. Schmidt, P. Soffer, E. Hunt, X. Franch, and R. Coletta, editors, *Proceedings of the 9th Workshop on Business Process Modelling, Development and Support (BPMDS)*, volume 335 of *CEUR Workshop Proceedings*, pages 10–18. CEUR-WS.org, 2008. (cited on p. 113)

[175] A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *12th IEEE International Enterprise Distributed Object Computing (EDOC 2008)*, pages 325–331. IEEE Computer Society, 2008. (cited on pp. 15 and 142)

[176] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986. (cited on pp. 229, 238, 239, and 260)

[177] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993. (cited on pp. 229, 238, 239, and 260)

[178] R. Krishnapuram and A. Joshi and L. Yi. A Fuzzy Relative of the k-medoids Algorithm with Application to Web Document and Snippet Clustering. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZIEEE)*, pages 1281–1286, 1999. (cited on p. 51)

[179] W. M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of American Statistical Association*, 66(336):846–850, 1971. (cited on p. 97)

[180] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In W. M. P. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer-Verlag, Berlin, 2003. (cited on pp. 42, 50, 67, 98, 124, 130, and 155)

[181] G. Regev, P. Soffer, and R. Schmidt. Taxonomy of Flexibility in Business Processes. In G. Regev, P. Soffer, and R. Schmidt, editors, *Proceedings of the 7th Workshop on Business Process Modelling, Development and Support (BPMDS)*, volume 236 of *CEUR Workshop Proceedings*, pages 90–93. CEUR-WS.org, 2006. (cited on p. 113)

[182] H. A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry.* Springer-Verlag, Berlin, 2003. (cited on p. 175)

[183] H. A. Reijers and J. Mendling. A Study Into the Factors That Influence the Understandability of Business Process Models. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(3):449–462, 2011. (cited on p. 98)

[184] E. S. Ristad and P. N. Yianilos. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998. (cited on pp. 74 and 78)

[185] S. Rogers. Big Data is Scaling BI and Analytics–Data Growth is About to Accelerate Exponentially–Get Ready. *Information Management-Brookfield*, 21(5):14, 2011. (cited on pp. 3 and 4)

[186] M. La Rosa and M. Dumas. Configurable Process Models: How To Adopt Standard Practices In Your Own Way? BPTrends Newsletter, November 2008. (cited on p. 324)

[187] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand. Exponentially Weighted Moving Average Charts for Detecting Concept Drift. *Pattern Recognition Letters*, 33 (2):191–198, 2012. (cited on p. 114)

[188] P. J. Rousseeuw. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. (cited on p. 96)

[189] I. Rouvellou and G. W. Hart. Automatic Alarm Correlation for Fault Identification. In *Proceedings of the Fourteenth Annual International Conference on Computer Communications (IEEE INFOCOM)*, pages 553–561. IEEE Computer Society, 1995. (cited on p. 229)

[190] A. Rozinat and W. M. P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. J. Bussler and A. Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006. (cited on p. 174)

[191] A. Rozinat and W. M. P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008. (cited on pp. 11, 156, 160, 168, and 200)

[192] W. J. Rudman, J. S. Eberhardt, W. Peirce, and S. Hart-Hester. Healthcare Fraud and Abuse. *Perspectives in Health Information Management*, 6, 2009. (cited on p. 228)

[193] N. Russel, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow Data Patterns. Technical Report FIT-TR-2004-01, Queensland University of Technology, 2004. (cited on p. 41)

[194] N. Russel, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow Resource Patterns. Technical report, Eindhoven University of Technology, 2004. (cited on p. 41)

[195] N. Russel, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical Report BPM-06-22, BPM-Center, 2006. (cited on p. 40)

[196] N. Russel, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Exception Handling Patterns in Process-Aware Information Systems. Technical Report BPM-06-04, BPM-Center, 2006. (cited on p. 41)

[197] S. Rusu. Discovery and Analysis of Field Service Engineer Process Using Process Mining. Master's thesis, Eindhoven University of Technology, 2010. (cited on p. 175)

[198] R. Sankaranarayanan and J.Ferlay. Worldwide Burden of Gynaecological Cancer: The Size of the Problem. *Best Practice & Research Clinical Obstetrics & Gynaecology*, 20 (2):207–225, 2006. (cited on p. 266)

[199] J. Schlimmer and R. Granger. Beyond Incremental Processing: Tracking Concept Drift. In Tom Kehler, editor, *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 502–507. Morgan Kaufmann, 1986. (cited on p. 114)

[200] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the Support of a High-dimensional Distribution. *Neural Computation*, 13(7): 1443–1471, 2001. (cited on pp. 236 and 260)

[201] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst. Process Flexibility: A Survey of Contemporary Approaches. In J. Dietz, A. Albani, and J. Barjis, editors, *Advances in Enterprise Engineering I*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer-Verlag, Berlin, 2008. (cited on pp. 113 and 115)

[202] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2004. ISBN 1584884401. (cited on pp. 122 and 253)

[203] M. Simonsen, T. Mailund, and C. N. S. Pedersen. Rapid Neighbor-Joining. In *Algorithms in Bioinformatics*, pages 113–122, 2008. (cited on p. 208)

[204] P. Smyth and R. M. Goodman. Rule Induction Using Information Theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. AAAI Press, 1991. (cited on p. 120)

[205] D. Sokol, G. Benson, and J. Tojeira. Tandem Repeats Over the Edit Distance. *Bioinformatics*, 23(2):e30–e35, 2007. (cited on pp. 47 and 75)

[206] M. Solé and J. Carmona. Region-based Foldings in Process Discovery. *IEEE Transactions on Knowledge and Data Engineering*, 2011. doi: http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.192. (cited on p. 141)

[207] M. Song and W. M. P. van der Aalst. Supporting Process Mining by Showing Events at a Glance. In K. Chari and A. Kumar, editors, *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145, 2007. (cited on p. 200)

[208] M. Song, C. W. Günther, and W. M. P. van der Aalst. Improving Process Mining with Trace Clustering. *Journal of the Korean Institute of Industrial Engineers*, 34(4): 460–469, 2008. (cited on pp. 79 and 80)

[209] M. Song, C. W. Günther, and W. M. P. van der Aalst. Trace Clustering in Process Mining. In D. Ardagna, M. Mecella, and J. Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 109–120. Springer-Verlag, Berlin, 2009. (cited on pp. 12, 79, 80, and 107)

[210] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. ASQ Quality Press, 2003. (cited on p. 229)

[211] J. W. Thornton and R. DeSalle. Gene Family Evolution and Homology: Genomics Meets Phylogenetics. *Annual Review of Genomics and Human Genetics*, 1:41–73, 2000. (cited on p. 324)

[212] A. De Toni and S. Tonchia. Performance Measurement Systems: Models, Characteristics and Measures. *International Journal of Operations & Production Management*, 21(1/2):46–70, 2001. (cited on p. 175)

[213] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Handling Local Concept Drift with Dynamic Integration of Classifiers: Domain of Antibiotic Resistance in Nosocomial Infections. In *19th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2006)*, pages 679–684, 2006. (cited on p. 114)

[214] E. Ukkonen. On-Line Construction of Suffix Trees. *Algorithmica*, 14(3):249–260, 1995. (cited on p. 56)

[215] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984. (cited on pp. 325 and 326)

[216] W. M. P. van der Aalst. Re-engineering Knock-out Processes. *Decision Support Systems*, 30(4):451–468, 2001. (cited on p. 111)

[217] W. M. P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering*, 10(3):198–211, 2005. (cited on p. 11)

[218] W. M. P. van der Aalst. Trends in Business Process Analysis: From Verification to Process Mining. In J. Cordeiro, J. Cardoso, and J. Filipe, editors, *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS)*, pages 12–22. Institute for Systems and Technologies of Information, Control and Communication, INSTICC, Medeira, 2007. (cited on p. 174)

[219] W. M. P. van der Aalst. Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. In R. Meersman, T. Dillon, and P. Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2010*, volume 6426 of *Lecture Notes in Computer Science*, pages 8–25. Springer-Verlag, Berlin, 2010. (cited on p. 284)

[220] W. M. P. van der Aalst. Intra– and Inter–Organizational Process Mining: Discovering Processes Within and Between Organizations. In P. Johannesson, J. Krogstie, and A. L. Opdahl, editors, *The Practice of Enterprise Modeling (PoEM)*, volume 92 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer-Verlag, Berlin, 2011. (cited on p. 284)

[221] W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer-Verlag New York Inc, 2011. (cited on pp. 4, 6, 9, 10, 15, 137, and 175)

[222] W. M. P. van der Aalst and A. K. A. de Medeiros. Process Mining and Security: Detecting Anamolous Process Executions and Checking Process Conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 2005. (cited on p. 11)

[223] W. M. P. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach.* The MIT Press, 2011. (cited on p. 40)

[224] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005. (cited on pp. 41, 48, and 111)

[225] W. M. P. van der Aalst and B. F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *Engineering and Deployment of Cooperative Information Systems*, volume 2480 of *Lecture Notes*

in Computer Science, pages 107–110. Springer-Verlag, Berlin, 2002. (cited on pp. 140, 174, and 175)

[226] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Workflow Modeling Using Proclets. In O. Etzion and P. Scheuermann, editors, *Cooperative Information Systems (CoopIS)*, volume 1901 of *Lecture Notes in Computer Science*, pages 198–209. Springer-Verlag, Berlin, 2000. (cited on pp. 144 and 322)

[227] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001. (cited on p. 322)

[228] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. (cited on pp. 18 and 40)

[229] W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. (cited on pp. 9, 52, 98, 137, 140, 174, and 325)

[230] W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DoA and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, Berlin, 2005. (cited on pp. 11, 18, and 201)

[231] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005. (cited on pp. 140 and 159)

[232] W. M. P. van der Aalst, B. F. van Dongen V. Rubin, E. Kindler, and C. W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. Technical Report BPM-06-30, BPM Center, 2006. (cited on p. 141)

[233] W. M. P. van der Aalst, N. Lohmann, M. La Rosa, and J. Xu. Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management*, volume 6336 of *Lecture Notes in Computer Science*, pages 95–111. Springer-Verlag, Berlin, 2010. (cited on pp. 133 and 324)

[234] W. M. P. van der Aalst, M. Pesic, and M. Song. Beyond Process Mining: From the Past to Present and Future. In B. Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer-Verlag, Berlin, 2010. (cited on p. 323)

[235] W. M. P. van der Aalst, K. M. van hee, J. M. van der Werf, and M. Verdonk. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *Computer*, 43(3):90–93, 2010. (cited on p. 11)

[236] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song. Time Prediction based on Process Mining. *Information Systems*, 36(2):450–475, 2011. (cited on p. 323)

[237] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. (cited on pp. 11 and 200)

[238] J. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. In K. M. van Hee and R. Valk, editors, *Applications and Theory of Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, Berlin, 2008. (cited on pp. 9, 137, 141, 159, and 169)

[239] B. F. van Dongen. *Process Mining and Verification.* PhD thesis, Eindhoven University of Technology, 2007. (cited on p. 141)

[240] B. F. van Dongen and A. Adriansyah. Process Mining: Fuzzy Clustering and Performance Visualization. In S. Rinderle-Ma, S. Sadiq, and Frank Leymann, editors, *Business Process Mangement Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 158–169. Springer-Verlag, Berlin, 2010. (cited on pp. 174 and 178)

[241] B. F. van Dongen and A. Adriansyah. Process Mining: Fuzzy Clustering and Performance Visualization. In S. Rinderle-Ma, S. Sadiq, and F. Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 158–169. Springer-Verlag, Berlin, 2010. (cited on pp. 11, 17, and 51)

[242] B. F. van Dongen and W. M. P. van der Aalst. EMiT: A Process Mining Tool. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets*, volume 3099 of *Lecture Notes in Computer Science*, pages 454–463. Springer-Verlag, Berlin, 2004. (cited on p. 140)

[243] B. F. van Dongen and W. M. P. van der Aalst. Multi-phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T. W. Ling, editors, *Conceptual Modeling–ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004. (cited on p. 141)

[244] B. F van Dongen and W. M. P. van der Aalst. A Meta Model for Process Mining Data. In J. Casto and E. Teniente, editors, *Proceedings of the CAiSE Workshops (EMOI-INTEROP Workshop)*, volume 2, pages 309–320, 2005. (cited on pp. 33 and 247)

[245] B. F. van Dongen and W. M. P. van der Aalst. Multi-phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordingation, Workflow and Business Process Management (PNCWB)*, pages 35–58, 2005. (cited on p. 141)

[246] B. F. van Dongen, A. K. A. de Medeiros, and L. Wen. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. In K. Jensen and W. M. P. van der Aalst, editors, *Transactions on Petri Nets and Other Models of Concurrency II*, volume 5460 of *Lecture Notes in Computer Science*, pages 225–242. Springer-Verlag, Berlin, 2009. (cited on pp. 9 and 137)

[247] Ministerie van Infrastructuur en Milieu. All-in-one permit for physical aspects: (omgevingsvergunning) in a nutshell, 2010. (cited on p. 284)

[248] M. van Leeuwen and A. Siebes. StreamKrimp: Detecting Change in Data Streams. In B. Goethals W. Daelemans and K. Morik, editors, *Machine Learning and Knowledge Discovery in Databases, (ECML/PKDD) Part I*, volume 5211 of *Lecture Notes in Computer Science*, pages 672–687. Springer-Verlag, Berlin, 2008. (cited on p. 114)

[249] G. M. Veiga and D. R. Ferreira. Understanding Spaghetti Models with Sequence Clustering for ProM. In S. Rinderle-Ma, S. Sadiq, and F. Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 92–103. Springer-Verlag, Berlin, 2010. (cited on pp. 80, 98, 99, and 105)

[250] J. J. C. L. Vogelaar, H. M. W. Verbeek, B. Luka, and W. M. P. van der Aalst. Comparing Business Processes to Determine the Feasibility of Configurable Models: A Case Study. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 50–61. Springer-Verlag, Berlin, 2012. (cited on p. 284)

[251] M. von den Driesch and T. Blickle. Operational, Tool-Supported Corporate Performance Management with the ARIS Process Performance Manager. In A. W. Scheer, W. Jost, H. Heß, and A. Kronz, editors, *Corporate Performance Management: ARIS in Practice*, pages 45–64. Springer-Verlag, Berlin, 2006. (cited on p. 174)

[252] I. Žliobaitė. Learning under Concept Drift: an Overview. Technical report, Faculty of Mathematics and Informatics, Vilnius University: Vilnius, Lithuania, 2009. (cited on p. 114)

[253] I. Žliobaitė and M. Pechenizkiy. Handling Concept Drift in Information Systems, 2010. `http://sites.google.com/site/zliobaite/CD_applications_2010.pdf`. (cited on p. 114)

[254] W. M. P. van der Aalst and V. Rubin and H. M. W. Verbeek and B. F. van Dongen and E. Kindler and C. W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010. (cited on p. 51)

[255] R. A. Wagner and M. J. Fisher. The String to String Correction Problem. *Journal of the ACM*, 21(1):168–173, 1974. (cited on p. 86)

[256] T. J. Walsh, M. L. Littman, and A. Borgida. Learning Web-Service Task Descriptions from Traces. *Web Intelligence and Agent Systems*, 2011. to appear. (cited on p. 326)

[257] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining Concept-drifting Data Streams Using Ensemble Classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM, 2003. (cited on p. 114)

[258] L. Wang and T. Jiang. On the Complexity of Multiple Sequence Alignment. *Journal of Computational Biology*, 1(4):337–348, 1994. (cited on p. 206)

[259] J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Assocication*, 58(301):236–244, 1963. (cited on p. 98)

[260] M. S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman & Hall/CRC, 2000. (cited on pp. 199, 201, and 204)

[261] B. Weber, S. Rinderle, and M. Reichert. Change Patterns and Change Support Features in Process-Aware Information Systems. In J. Krogstie, A. Opdahl, and G. Sindre, editors, *Advanced Information Systems Engineering (CAiSE)*, volume 4495 of *Lecture Notes in Computer Science*, pages 574–588. Springer-Verlag, Berlin, 2007. (cited on pp. 113, 115, and 324)

[262] P. Weber. Research Progress Report 3-Thesis Proposal: Machine Learning in Process Mining. Technical report, School of Computer Science, Univeristy of Birmingham, 2010. `http://www.cs.bham.ac.uk/~pxw869/papers/RSMG/RSMG3thesisproposal.pdf`. (cited on p. 326)

[263] A. Weigel and F. Fein. Normalizing the Weighted Edit Distance. In *Proceedings of the 12th International Conference on Pattern Recognition, Conference-B: Computer Vision and Image Processing*, volume 2, pages 399–402. IEEE Computer Soceity, 1994. (cited on p. 92)

[264] A. J. M. M. Weijters and W. M. P. van der Aalst. Rediscovering Workflow Models From Event-based Data Using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003. (cited on pp. 9, 52, 97, 101, 103, 137, 140, 159, 268, 286, 289, 295, and 307)

[265] L. Wen, J. Wang, W. M. P. van der Aalst, Z. Wang, and J. Sun. A Novel Approach for Process Mining Based on Event Types. Technical Report BETA Working Paper Series, WP 118, Eindhoven University of Technology, 2004. (cited on p. 140)

[266] L. Wen, J. Wang, and J. Sun. Detecting Implicit Dependencies Between Tasks from Event Logs. In X. Zhou, J. Li, H. T. Shen, M. Kitsuregawa, and Y. Zhang, editors, *Frontiers of WWW Research and Development–APWeb 2006*, volume 3841 of *Lecture Notes in Computer Science*, pages 591–603. Springer-Verlag, Berlin, 2006. (cited on p. 140)

[267] G. Widmer and M. Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23(1):69–101, 1996. (cited on pp. 109 and 114)

[268] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java Implementation. In *Proceedings of the ICONIP/ANZIIS/ANNES Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*, pages 192–196, 1999. (cited on p. 260)

[269] M. T. Wynn, D. Edmond, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets*, volume 3536 of *Lecture Notes in Computer Science*, pages 1266–1269. Springer-Verlag, Berlin, 2005. (cited on p. 41)

[270] J. Xia. Automatic Determination of Graph Simplification Parameter Values for Fuzzy Miner. Master's thesis, Eindhoven University of Technology, 2010. (cited on p. 255)

[271] W. S. Yang and S. Y. Hwang. A Process Mining Framework for the Detection of Healthcare Fraud and Abuse. *Expert Systems with Applications*, 31(1):56–68, 2006. (cited on pp. 11 and 228)

[272] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy. SherLog: Error Diagnosis by Connecting Clues from Run-time Logs. In J. C. Hoe and V. S. Adve, editors, *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 143–154. ACM, 2010. (cited on p. 229)

[273] L. Yujian and L. Bo. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007. (cited on p. 92)

Summary

Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

Modern information systems supporting operational processes of organizations record events (data) capturing the footprints of process executions. Such data is generated not only by traditional business information systems but also through numerous other sources[1]. The amount and the rate at which data is collected by organizations today are growing faster than ever. Intelligent use of this voluminous data is perceived to be one of the key challenges in the future for organizations to innovate, grow, and retain their competitive edge.

Process mining is an emerging discipline that aims at extracting non-trivial process related knowledge and interesting insights from event logs, e.g., to automatically discover process models. Such knowledge can be used to monitor and improve real processes. Remarkable success stories have been reported on the applicability of process mining based on event logs from real-life workflow management/information systems. Today, we are at the cross roads of an increasing number of unprecedented domains and new applications willing to apply and adopt process mining. Process mining is being looked at even in applications that are atypical of workflow systems. Analysis of event logs of high-tech systems such as X-ray machines and CT scanners (medical systems), copiers and printers, mission-critical information systems, etc., are a few examples illustrating this trend.

Processes executed on high-tech systems tend to be flexible and/or less-structured while the event logs from these systems tend to be *fine-granular*, *heterogenous*, and *voluminous*. We refer to logs with the above characteristics as large scale event logs. These characteristics pose new challenges to process mining. For example, contemporary process discovery algorithms have problems in dealing with fine-grained event logs and less-structured processes, and generate spaghetti-like process models that are hard to comprehend. *This thesis focusses on enabling process mining for large scale event logs.* More specifically, this thesis addresses the following research questions:

- how do we deal with fine-granular event logs and less-structured processes?
- how do we deal with heterogeneity in event logs?
- how do we deal with process changes? and

[1]Big Data: The Next Frontier for Innovation, Competition and Productivity, McKinsey Global Institute, May 2011.

- how do we enhance support for process diagnostics?

In this thesis, we advocate that the problems arising in analyzing large scale event logs can be tackled from two fronts: (i) through *event log simplification* and (ii) through *advancements in process mining*. We develop an approach to form abstractions over events by exploiting the common execution patterns manifested in an event log. Such patterns typically carry a strong conceptual relationship (of domain significance) between the events involved in the pattern. Event logs can be simplified by replacing the low-level events with abstract activities thereby mitigating the problem of fine granular events. Using this as a basis, we propose a two-phase approach to process discovery that enables the discovery of hierarchical process models (process maps). This approach mitigates the problem of dealing with less-structured processes by allowing the discovery of navigable process maps with multiple-levels of hierarchy and context-dependent views.

In this thesis, we show that process mining results on heterogenous event logs can be improved by first partitioning the event log into subsets of homogenous cases (*trace clustering*) and analyzing these subsets independently. We highlight the significance of considering contexts during such partitioning and propose context-aware approaches to trace clustering. Furthermore, we introduce the topic of *concept drift* to deal with process changes and propose techniques to detect points of change in an event log. Detection of such change points enables the selection of cases and putting the analysis results in perspective to process variants.

Another major contribution of this thesis is in enhancing the support for process diagnostics. Process diagnostics, which encompasses process conformance checking, auditing, process performance analysis, anomaly detection, diagnosis, inspection of interesting patterns and the like, is gaining prominence in recent years. However, lion's share of process mining research has been devoted to control-flow discovery. In this thesis, we try to bridge this gap and propose several techniques for process diagnostics. We propose a means of replaying an event log onto process maps so that process performance can be measured and bottlenecks identified. We develop a trace visualization technique, called as *trace alignment*, that aligns the traces in such a way that event logs can be explored easily. Trace alignment extends the scope of conformance checking to also cover the direct inspection of traces. We show that trace alignment is extremely helpful in answering a variety of diagnostic questions such as *where do process instances deviate?* and *are there any common patterns of execution in the process instances?*. We also propose techniques for discovering signature patterns that discriminate between different classes of behavior in event logs, e.g., patterns that discriminate fraudulent insurance claims from normal claims.

The algorithms and techniques described in this thesis are supported by concrete implementations as plug-ins in the ProM framework[2] and have been applied on real-life case studies. We also show that most of the techniques presented in this thesis are scalable, which is one of the desirable aspects in analyzing large scale event logs.

[2]see www.processmining.org to download and to learn more about ProM.

Samenvatting

Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

Van vrijwel alle processen die tegenwoordig door (informatie)systemen ondersteund worden, wordt continu opgeslagen wat er precies gebeurt in de vorm van event logs. Het volume van die data, opgeslagen door traditionele workflowsystemen, maar ook door vele andere systemen, groeit sneller dan ooit[3]. Een van de grootste uitdagingen voor organisaties om te innoveren, te groeien en hun concurrentiepositie te versterken is om intelligent gebruik te maken van de beschikbare data.

Het verkrijgen van niet-triviale, procesgerelateerde inzichten door middel van de analyse van eventdata is een relatief nieuwe discipline genaamd "process mining". Met behulp van de verkregen inzichten, bijvoorbeeld in de vorm van procesmodellen, kunnen operationele processen bewaakt en verbeterd worden. De toepassing van process mining op eventdata van informatiesystemen heeft in de praktijk geleid tot verbluffende successen. Ook de toepassing van process mining op eventdata die niet direct afkomstig is van informatiesystemen, maar van allerlei andere high-tech systemen, zoals röntgenapparaten, CT scanners, kopieermachines, printers, etc., heeft onlangs een grote vlucht genomen.

Processen die uitgevoerd worden in high-tech systemen zijn vaak erg flexibel en ongestructureerd, wat ertoe leidt dat de logs *fijnmazig*, *heterogeen* en *erg omvangrijk* worden. In dit proefschrift worden zulke logs aangeduid met de term "large scale event logs". Dergelijke karakteristieken zorgen voor extra uitdagingen op het gebied van process mining. Zo kunnen de meeste technieken bijvoorbeeld niet omgaan met de combinatie van fijnmazige logs en ongestructureerde processen en vaak leiden deze technieken dan ook tot onleesbare modellen. *Dit proefschrift beschrijft technieken om process mining toe te kunnen passen op large scale event logs.* Dit wordt beschreven aan de hand van de volgende onderzoeksvragen:

- Hoe moeten we omgaan met fijnmazige, ongestructureerde processen?
- Hoe moeten we omgaan met heterogene event logs?
- Hoe moeten we omgaan met veranderingen in processen? en
- Hoe kunnen we mogelijkheden tot procesbewaking uitbreiden?

[3]Big Data: The Next Frontier for Innovation, Competition and Productivity, McKinsey Global Institute, May 2011.

De problemen die voortkomen uit het omgaan met fijnmazige, ongestructureerde processen worden in dit proefschrift van twee kanten benaderd, namelijk via (i) *versimpeling van logs* en (ii) door middel van *uitbreiding van process mining technieken.* Voor de versimpeling van logs wordt een aanpak gepresenteerd die patronen vindt in events. Events binnen deze patronen blijken vaak sterk gerelateerd wanneer gekeken wordt naar het systeem waar de log vandaan komt en door de events in de log te vervangen door de patronen kunnen bestaande process mining technieken gebruikt worden op de ontstane log. Met dit als basis wordt in dit proefschrift een gefaseerde aanpak gepresenteerd die in staat is om hiërarchische modellen, zogenaamde "process maps", te verkrijgen uit een large scale event log. Door iteratieve toepassing van deze techniek, kunnen ongestructureerde processen inzichtelijk gemaakt worden op verschillende, contextafhankelijke niveaus.

Om process mining toe te kunnen passen op heterogene event logs, wordt in dit proefschrift een *clustering* aanpak gepresenteerd. Hierbij worden homogene delen van de event log uitgesplitst en onafhankelijk van elkaar geanalyseerd. Bij het clusteren van delen van de log is het van groot belang om de context waarin events plaatsvonden mee te nemen en de technieken die gepresenteerd worden zijn dan ook *context aware.* Aangezien veranderingen in een proces, ook veranderingen in de context van events met zich meebrengen, wordt de notie van *concept drift* geïntroduceerd. Hiermee kan in de log aangewezen worden wanneer een proces veranderde, waardoor de log opnieuw gesplitst kan worden in verschillende delen voor verdere analyse.

Behalve aan het ontdekken van procesmodellen wordt in dit proefschrift ook veel aandacht besteed aan de mogelijkheden tot procesbewaking. Hierbij gaat het om zaken als het in kaart brengen van prestaties van een proces, het identificeren van afwijkingen ten opzichte van modellen, alsmede het (tijdig) detecteren van mogelijk problematische afwijkingen van de normale gang van zaken. Hoewel de meeste process mining technieken weinig aandacht besteden aan procesbewaking, is dit iets wat vooral de laatste tijd steeds belangrijker is geworden. Door de event log na te spelen op de ontdekte process maps, kan inzichtelijk gemaakt worden waar bottlenecks zitten en hoe de werkelijkheid afwijkt van het model. Daarnaast wordt een uitlijntechniek gepresenteerd genaamd *trace alignment* waarmee instanties van een proces snel en gemakkelijk met elkaar vergeleken kunnen worden. Met deze techniek is het bijvoorbeeld mogelijk om snel te zien waar bepaalde instanties afwijkingen vertonen, maar ook of bepaalde afwijkingen vaak voorkomen, bijvoorbeeld in geval van frauduleuze handelingen. Ook bij deze techniek is het weer mogelijk de log te splitsen om zo de afwijkende zaken verder te analyseren.

Alle algoritmen en technieken die in dit proefschrift beschreven worden zijn geïmplementeerd in het ProM framework[4]. Daarnaast zijn alle technieken toegepast op echte processen om de relevantie en schaalbaarheid in de praktijk te bewijzen.

[4]Zie www.processmining.org voor meer informatie over de tool en om deze te installeren.

Acknowledgments

Although my interest in pursuing a PhD dates back to 2004, I made a serious attempt at it only in late 2007 when I was working as a researcher in Accenture Technology Labs (ATL), Bangalore, India. My work at ATL led me to explore the area of service-oriented architecture. One thing led to another and I ended up reading a lot of papers related to workflow and process mining. Although my initial preference was to do a PhD in one of the top universities in the U.S.A., I landed up in TU/e because I found the topic of process mining interesting and the papers that most delighted me in this area had Wil's name on them.

I started my PhD in April 2008 under the supervision of Wil. Throughout these four years, Wil has enlightened me through his wealth of knowledge and deep intuitions. Wil has been a role model and his joy and enthusiasm for research has been contagious and a source of inspiration for me. Wil has always encouraged and challenged me to grow as a researcher and an independent thinker whilst giving me all the freedom to work in my own way and pushed me to deliver nothing less than my best efforts. This thesis would not have taken its current shape without his guidance and rigorous comments. The anecdotes and metaphors that he shared with me have added new perspectives to my outlook and thinking. Thank you, Wil, for everything you have done for me. I will always remain indebted to you.

I would also like to thank my co-promotor, Boudewijn, for his help and feedback and also for translating the Samenvatting in Dutch. I would also like to thank Eric for his help on ProM related matters. Together, Boudewijn and Eric have always been there with an open-door policy to help clarify any questions and issues that I had with ProM. They have made development in ProM an enjoyable experience.

I would like to thank my reading committee members, prof. Marlon Dumas, prof.dr.ir. Kees van Hee, and dr. Gianluigi Greco for their time, interest, and helpful comments. I would also like to thank the other members of my defense committee, prof. Arno Siebes, prof. Jan Vanthienen, and prof.dr. Arjeh Cohen.

I would also like to thank Philips Healthcare for their generous funding of my PhD project. Several people at Philips have extended their help and support in the project. I thank John, Bert, Marcel, Pieter, Peter, Eugene, Richard, Krzysztof, Guillaume, Wim, Kenny, Janwillem, Saskia, and Frans for their support and interest in my work. I also extend my thanks to Eric and Bart for their support. Special thanks go out to Eugene and Krzysztof, with whom I had the opportunity to work with more closely. A very special thanks goes to Richard for all his help and collaboration

in the project over the last two years and for reviewing the thesis. Richard, you have been a pleasure to work with and have always been there willing to help in spite of all the pressures and workload that you had. I appreciate all that you have done for me.

I have had the great fortune to cross paths with many dedicated teachers and benefit from their wisdom, encouragement, and friendship. Of them, three professors have had a profound impact on me. I owe my thanks to Prof. J. Ramakrishna for his inspiring words and advice which have been instrumental in moulding my career. Sir, I know that you are't too happy with me for not considering to do my PhD at IISc but I am sure that you will appreciate my decision after seeing this thesis. I am indebted to prof. P.C.P. Bhatt for being a mentor and for his help, advice, and encouragement over the years. Sir, I always enjoyed the discussions with you and they have been a learning experience for me. I express my sincere thanks to prof. G. Nagaraja for introducing me to the field of machine learning and pattern recognition, for advising me in my Master's thesis, and for extending his help ever since in all my pursuits. Thank you sir. You have played a vital role in making me a good researcher. I also express my gratitude to prof. Jayarami Reddy, prof. Pushpak Bhattacharyya, prof. Krithi Ramamritham, prof. Balaji, Dr. K.P. Ramesh, and Dr. S.H. Srinivasan. I also wish to extend my sincere appreciation for all the teachers who have educated me throughout my career.

I would also like to thank Dr. Lin L. Chase (Principal at SecondLook and formerly, Director of Accenture Technlogy Labs, India) for her encouragement and support throughout my tenure at ATL and subsequently during my PhD. Ever since our first meeting in April 2006, Lin has always encouraged me to pursue my PhD and has extended her support in numerous ways during my stint at ATL and beyond as well. Lin, I have always looked up to you as a source of inspiration and a role-model. Thank you for all your help and support. I also wish to extend my thanks to Kishore Swaminathan, Edy Liongosary, Andrew Fano, Alex Kass, Scott Kurth, and Sanjay Mathur for fully extending their support when I decided to leave ATL to pursue my PhD.

I also extend my thanks to my friends and colleagues at ATL, Anindita, Anitha, Ashwin, Deb, Gurdeep, Lakshmi, Lissy, Nithya, Pyush, Rajiv, Roshni, Sangeetha, Santonu, Venkatesh, Vibhu, and Vishwa. Special thanks go to Vikrant, Subramanya Prasad, Atul, and Suma for their interest in my progress. Thanks to all of you for making my stay in ATL one of the most memorable ones.

Many thanks are due to Riet and Ine who have been wonderful and took care of all the administrative things with smile. I also extend my thanks to my colleagues at TU/e, Ana Karla, Anne, Arjan, Arya, Carmen, Christian Günther, Christian Stahl, Debjyoti, Dennis, Dirk, Fabrizio, Faisal, Hajo, Helen, Irene, Jan Martijn, Joos, Joyce, Maja, Massimiliano, Michael, Minsoek, Natalia, Ronny, and Ton, who have been friendly and cheerful. I cherish the discussions that I had with some of you and thank you for showing your interest in my work. Special thanks go to Jan Martijn for guiding me through the formalities of the thesis and defense preparation and Hajo for making some of my conference visits a memorable one with his witty humor. I also thank Indrė and Mykola for their collaboration on concept drift. I also extend my thanks to all the members of the DH group. My thanks also go to Stefania and Fan Liu, two of my master's students, for their help and to Danny for the fruitful discussions that we had.

My time in Eindhoven was made enjoyable in large part due to the many friends. The very presence of Subbu around made me feel like I am at home and provided us an opportunity to go back in time and discuss the experiences that we shared together time and again. Special thanks go to Nagaraj and Pushpa for their support and for having me over for innumerable lunches and dinner. You have made me not miss most of the festivals. I also thank my friends Anirban and Sushmita for their help and support in Eindhoven and for hosting me in Cambridge during my visit. My thanks also go to Surendra for the good times that we had and our memorable trips. My thanks also go out to my friends, Prem, Chaitanya, Sravanthi, Poornima Manjunath, Sunil, Mallikarjun, Vijay, Naveen, Jaybharath, Sarath, and Ravikanth in India and elsewhere across the globe.

I would also like to thank the ARC Centre of Excellence in Plant Energy Biology, The University of Western Australia, Crawley, and its director, prof. Ian Small, for permitting me to use the picture on the cover page.

I would also like to express my gratitude to my alma maters, Indian Institute of Technology Bombay (IIT Bombay) and Indian Institute of Science (IISc), for providing me with an outstanding educational base and an opportunity to learn from the many distinguished teachers. The ambience and environment at IISc played a significant role in my transformation and my reverence for it is inexplicable.

My deepest gratitude goes to my beloved parents, R. Prabhakara Naidu and R.P. Vimala Devi, for their relentless sacrifices, unconditional love and support throughout my life. This thesis wouldn't have been possible without them. They have imbibed in me the importance of education and provided me all that they can towards realizing this. Thank you amma and nanna. This thesis is dedicated to you. I also want to thank my brother, Dr. Sreekanth Chakradhar, for his help and support. The numerous books that you have bought for me during my school and graduate days have all played a role in enriching my fundamentals. I also want to thank my sister, Dr. Jaya Prasanthi, for her encouragement and support. My brother-in-law, Chandrasekhar Allu, has also become an important person in my world. You both have helped me in many ways and made my conference travels to the U.S.A. a memorable one. Also, thanks very much for the timely gifts: laptop, hard discs, iphone, etc., and many more. A very special thanks to the little princess in our family, my niece Divija, for the bundles of joy that she brings in.

Eindhoven, The Netherlands — R.P. Jagadeesh Chandra Bose
March 2012

Curriculum Vitae

R.P. Jagadeesh Chandra Bose (JC) was born on April 26, 1981 in Chittoor, India, where he also grew up. From 1998 to 2002, JC studied his Bachelor of Technology in Computer Science and Engineering at G. Pulla Reddy Engineering College, Kurnool, India. From 2002 to 2004, JC studied his Master of Technology in Computer Science and Engineering at the Indian Institute of Technology Bombay (IIT Bombay). The title of his dissertation is "Multi-strategy Learning and KBANN Performance". He has obtained various awards for excellence in academics, which include Gold medal for topping in under-graduation, National Merit Scholarship, Misys Charitable Foundation Scholarship, and Young Engineering Fellowship.

Subsequent to his graduation, JC worked as a researcher at Philips Research, Bangalore, India and later at Accenture Technology Labs, Bangalore, India. During his stint in the industry, he has executed projects in the areas of software engineering, healthcare, and knowledge management and has applied for five patent filings.

In April 2008, JC moved to Eindhoven to start his doctoral studies in the Department of Mathematics and Computer Science at Eindhoven University of Technology under the supervision of prof.dr.ir. Wil M.P. van der Aalst and the co-supervision of dr.ir. B.F. van Dongen. His project, funded by Philips Healthcare, is concerned with the application of process mining in the analysis of event logs from X-ray machines. His work has been published in leading journals and conferences/workshops. His research received the best student paper award at the 8th International Conference on Business Process Management 2010 and has won the First Business Process Intelligence Challenge 2011. His doctoral thesis, defended on 14 May 2012, is titled "Process Mining in the Large: Preprocessing, Discovery, and Diagnostics".

JC's research interests include machine learning, pattern recognition, data mining, process mining, business process management, bioinformatics, and computational neuroscience. As of April 2012, JC will be working as a PostDoc at Eindhoven University of Technology. He can be reached at jcbose@gmail.com.

Index

SIKS Dissertations

1998

1998-1 Johan van den Akker (CWI) DEGAS - An Active, Temporal Database of Autonomous Objects
1998-2 Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information
1998-3 Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspectives
1998-4 Dennis Breuker (UM) Memory versus Search in Games
1998-5 E.W.Oskamp (RUL) Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
1999-2 Rob Potharst (EUR) Classification using decision trees and neural nets
1999-3 Don Beal (UM) The Nature of Minimax Search
1999-4 Jacques Penders (UM) The practical Art of Moving Physical Objects
1999-5 Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
1999-6 Niek J.E. Wijngaards (VU) Re-design of compositional systems
1999-7 David Spelt (UT) Verification support for object database design
1999-8 Jacques H.J. Lenting (UM) Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-1 Frank Niessink (VU) Perspectives on Improving Software Maintenance
2000-2 Koen Holtman (TUE) Prototyping of CMS Storage Management
2000-3 Carolien M.T. Metselaar (UVA) Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.
2000-4 Geert de Haan (VU) ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM) Knowledge-based Query Formulation in Information Retrieval.
2000-6 Rogier van Eijk (UU) Programming Languages for Agent Communication
2000-7 Niels Peek (UU) Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coupâ (EUR) Sensitivity Analyis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI) Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI) Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI) Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU) Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU) Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA) Learning as problem solving
2001-4 Evgueni Smirnov (UM) Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU) Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU) Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU) Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU) A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9 Pieter Jan 't Hoen (RUL) Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10 Maarten Sierhuis (UvA) Modeling and Simulating Work Practice; BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11 Tom M. van Engers (VUA) Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU) Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT) Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT) Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU) The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU) The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL) Applied legal epistemology; Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI) Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU) Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel(KUB) Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM) Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU) Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (Uva) Processing XML in Database Systems
2002-13 Hongjing Wu (TUE) A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU) Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT) Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU) The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA) Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU) Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU) Modal Action Logics for Reasoning About Reactive Systems
2003-03 Martijn Schuemie (TUD) Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04 Milan Petkovic (UT) Content-Based Video Retrieval Supported by Database Technology
2003-05 Jos Lehmann (UVA) Causation in Artificial Intelligence and Law - A modelling approach
2003-06 Boris van Schooten (UT) Development and specification of virtual environments
2003-07 Machiel Jansen (UvA) Formal Explorations of Knowledge Intensive Tasks
2003-08 Yongping Ran (UM) Repair Based Scheduling
2003-09 Rens Kortmann (UM) The resolution of visually guided behaviour
2003-10 Andreas Lincke (UvT) Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11 Simon Keizer (UT) Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12 Roeland Ordelman (UT) Dutch speech recognition in multimedia information retrieval
2003-13 Jeroen Donkers (UM) Nosce Hostem - Searching with Opponent Models
2003-14 Stijn Hoppenbrouwers (KUN) Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15 Mathijs de Weerdt (TUD) Plan Merging in Multi-Agent Systems
2003-16 Menzo Windhouwer (CWI) Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17 David Jansen (UT) Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18 Levente Kocsis (UM) Learning Search Decisions

2004

2004-01 Virginia Dignum (UU) A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02 Lai Xu (UvT) Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU) A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04 Chris van Aart (UVA) Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR) Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD) The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM) Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek(UM) Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiäle gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU) For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA) Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU) Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT) Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT) Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU) Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15 Arno Knobbe (UU) Multi-Relational Data Mining
2004-16 Federico Divina (VU) Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM) Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA) Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT) Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode) Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA) Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM)) AI techniques for the game of Go
2005-03 Franc Grootjen (RUN) A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT) Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UVA) Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM) Adaptive Game AI
2005-07 Flavius Frasincar (TUE) Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE) A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU) Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UVA) Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU) Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12 Csaba Boer (EUR) Distributed Simulation in Industry
2005-13 Fred Hamburg (UL) Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU) Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15 Tibor Bosse (VU) Analysis of the Dynamics of Cognitive Processes
2005-16 Joris Graaumans (UU) Usability of XML Query Languages
2005-17 Boris Shishkov (TUD) Software Specification Based on Re-usable Business Components
2005-18 Danielle Sent (UU) Test-selection strategies for probabilistic networks
2005-19 Michel van Dartel (UM) Situated Representation
2005-20 Cristina Coteanu (UL) Cyber Consumer Law, State of the Art and Perspectives
2005-21 Wijnand Derks (UT) Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE) Foundations of B2B Electronic Contracting
2006-02 Cristina Chisalita (VU) Contextual issues in the design and use of information technology in organizations
2006-03 Noor Christoph (UVA) The role of metacognitive skills in learning to solve problems
2006-04 Marta Sabou (VU) Building Web Service Ontologies
2006-05 Cees Pierik (UU) Validation Techniques for Object-Oriented Proof Outlines
2006-06 Ziv Baida (VU) Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling
2006-07 Marko Smiljanic (UT) XML schema matching – balancing efficiency and effectiveness by means of clustering
2006-08 Eelco Herder (UT) Forward, Back and Home Again - Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM) Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU) Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT) Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU) Interactivation - Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU) Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU) Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU) CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU) Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU) User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA) Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU) Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT) Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN) Aptness on the Web
2006-22 Paul de Vrieze (RUN) Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU) Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU) Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU) Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlovic (UT) Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI) Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA) Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT) Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG) Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU) Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU) Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL) Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA) Applied Text Analytics for Blogs
2007-07 Natasa Jovanovic' (UT) To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU) Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU) Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU) Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11 Natalia Stash (TUE) Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN) Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT) Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM) Context-Based Image Analysis
2007-15 Joyca Lacroix (UM) NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU) Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU) Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT) On the development an management of adaptive business collaborations
2007-19 David Levy (UM) Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU) Customer Configuration Updating in a Software Supply Network
2007-21 Karianne Vermaas (UU) Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
2007-22 Zlatko Zlatev (UT) Goal-oriented design of value and process models from patterns
2007-23 Peter Barna (TUE) Specification of Application Logic in Web Information Systems
2007-24 Georgina Ramírez Camps (CWI) Structural Features in XML Retrieval
2007-25 Joost Schalken (VU) Empirical Investigations in Software Process Improvement

2008

2008-01 Katalin Boer-Sorbán (EUR) Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
2008-02 Alexei Sharpanskykh (VU) On Computer-Aided Methods for Modeling and Analysis of Organizations
2008-03 Vera Hollink (UVA) Optimizing hierarchical menus: a usage-based approach
2008-04 Ander de Keijzer (UT) Management of Uncertain Data - towards unattended integration
2008-05 Bela Mutschler (UT) Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
2008-06 Arjen Hommersom (RUN) On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
2008-07 Peter van Rosmalen (OU) Supporting the tutor in the design and support of adaptive e-learning
2008-08 Janneke Bolt (UU) Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU) The paradox of the guided user: assistance can be counter-effective
2008-10 Wauter Bosma (UT) Discourse oriented summarization
2008-11 Vera Kartseva (VU) Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN) A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA) Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT) Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT) The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.
2008-16 Henriette van Vugt (VU) Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD) Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM) Adaptive Active Vision
2008-19 Henning Rode (UT) From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20 Rex Arendsen (UVA) Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.
2008-21 Krisztian Balog (UVA) People Search in the Enterprise
2008-22 Henk Koning (UU) Communication of IT-Architecture
2008-23 Stefan Visscher (UU) Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU) Using background knowledge in ontology matching
2008-25 Geert Jonker (UU) Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26 Marijn Huijbregts (UT) Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU) Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN) On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT) Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
2008-30 Wouter van Atteveldt (VU) Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
2008-31 Loes Braun (UM) Pro-Active Medical Information Retrieval
2008-32 Trung H. Bui (UT) Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
2008-33 Frank Terpstra (UVA) Scientific Workflow Design; theoretical and practical issues
2008-34 Jeroen de Knijf (UU) Studies in Frequent Tree Mining
2008-35 Ben Torben Nielsen (UvT) Dendritic morphologies: function shapes structure

2009

2009-01 Rasa Jurgelenaite (RUN) Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT) A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU) Understanding Classification
2009-07 Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN) Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UVA) Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UVA) Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universität zu Berlin) Operating Guidelines for Services
2009-13 Steven de Jong (UM) Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA) Ontology Representation - Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT) New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT) Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI) Armada, An Evolving Database System
2009-19 Valentin Robu (CWI) Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU) Adjustable Autonomy: Controling Influences on Decision Making
2009-21 Stijn Vanderlooy (UM) Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT) Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU) Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VUA) Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI) RAM: Array Database Management through Relational Mapping
2009-26 Fernando Koch (UU) An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU) Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT) Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT) Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI) Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA) A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT) How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN) Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universität zu Berlin) Service Substitution – A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT) Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT) Digital Analysis of Paintings
2009-42 Toine Bogers (UvT) Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT) Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT) Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU) Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA) Querying XML: Benchmarks and Recursion

2010

2010-01 Matthijs van Leeuwen (UU) Patterns that Matter
2010-02 Ingo Wassink (UT) Work flows in Life Science
2010-03 Joost Geurts (CWI) A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT) Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT) Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT) Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT) Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL) Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL) A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL) Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD) The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU) Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN) High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU) Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT) Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD) Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU) Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU) Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA) People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT) Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT) Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI) End-user Support for Access to
Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU) The Logical Structure of Emotions
2010-24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU) Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI) XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL) Automatisch contracteren
2010-28 Arne Koopman (UU) Characteristic Relational Patterns
2010-29 Stratos Idreos(CWI) Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT) Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA) Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT) An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT) Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU) Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification
2010-37 Niels Lohmann (TUE) Correctness of services and their composition
2010-38 Dirk Fahland (TUE) From Scenarios to Components
2010-39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM) Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach
2010-43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies
2010-44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
2010-45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services
2010-46 Vincent Pijpers (VU) e3alignment: Exploring Inter-Organizational Business-ICT Alignment
2010-47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples
2010-48 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets
2010-49 Jahn-Takeshi Saito (UM) Solving difficult game positions
2010-50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources 2010-52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
2010-53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

2011

2011-01 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2011-02 Nick Tinnemeier (UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
2011-03 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU) Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms
2011-05 Base van der Raadt (VU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
2011-06 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage
2011-07 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction
2011-08 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues
2011-09 Tim de Jong (OU) Contextualised Mobile Media for Learning
2011-10 Bart Bogaert (UvT) Cloud Content Contention
2011-11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective
2011-12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining
2011-13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling
2011-14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets
2011-15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval
2011-16 Maarten Schadd (UM) Selective Search in Games of Different Complexity
2011-17 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18 Mark Ponsen (UM) Strategic Decision-Making in complex games
2011-19 Ellen Rusman (OU) The Mind ' s Eye on Personal Profiles
2011-20 Qing Gu (VU) Guiding service-oriented software engineering - A view-based approach
2011-21 Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems
2011-22 Junte Zhang (UVA) System Evaluation of Archival Description and Access
2011-23 Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media
2011-24 Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
2011-25 Syed Waqar ul Qounain Jaffry (VU)) Analysis and Validation of Models for Trust Dynamics
2011-26 Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
2011-27 Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns
2011-28 Rianne Kaptein(UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE) Discrimination-aware Classification
2011-30 Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions
2011-31 Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32 Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science
2011-33 Tom van der Weide (UU) Arguing to Motivate Decisions
2011-34 Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35 Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training
2011-36 Erik van der Spek (UU) Experiments in serious game design: a cognitive approach
2011-37 Adriana Burlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
2011-38 Nyree Lemmens (UM) Bee-inspired Distributed Optimization
2011-39 Joost Westra (UU) Organizing Adaptation using Agents in Serious Games
2011-40 Viktor Clerc (VU) Architectural Knowledge Management in Global Software Development
2011-41 Luan Ibraimi (UT) Cryptographically Enforced Distributed Data Access Control
2011-42 Michal Sindlar (UU) Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU) Process Improvement through Software Operation Knowledge
2011-44 Boris Reuderink (UT) Robust Brain-Computer Interfaces
2011-45 Herman Stehouwer (UvT) Statistical Language Models for Alternative Sequence Selection
2011-46 Beibei Hu (TUD) Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
2011-47 Azizi Bin Ab Aziz(VU) Exploring Computational Models for Intelligent Support of Persons with Depression
2011-48 Mark Ter Maat (UT) Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
2011-49 Andreea Niculescu (UT) Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

2012-01 Terry Kakeeto (UvT) Relationship Marketing for SMEs in Uganda
2012-02 Muhammad Umair(VU) Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
2012-03 Adam Vanya (VU) Supporting Architecture Evolution by Mining Software Repositories
2012-04 Jurriaan Souer (UU) Development of Content Management System-based Web Applications
2012-05 Marijn Plomp (UU) Maturing Interorganisational Information Systems
2012-06 Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks
2012-07 Rianne van Lambalgen (VU) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
2012-08 Gerben de Vries (UVA) Kernel Methods for Vessel Traject
2012-09 Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms
2012-10 David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment
2012-11 J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12 Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems

The goal of process mining is to extract non-trivial process related knowledge and interesting insights from event logs. Today, there is an unprecedented growth of data from a wide variety of sources and systems across many domains and applications. The opportunities of data collection is going beyond traditional information systems to also include sensor and machine data, e.g., high-tech systems such as X-ray machines, wafer scanners, copiers and printers, etc. Although the availability of large amounts of event data is an important enabler for process mining, the volume of data and complexity of processes creates many challenges, e.g., traditional process discovery techniques uncover spaghetti-like incomprehensible models. This PhD thesis addresses the following three main challenges:

1. Dealing with less-structured processes
2. Dealing with process changes
3. Provisions for process diagnostics

This PhD thesis proposes various new process mining techniques addressing the above challenges. All of these techniques have been implemented in ProM and evaluated using a variety of case studies.

ISBN: 978-90-386-3118-9