Aligning observed and modeled behavior

Aligning Observed and Modeled Behavior

Arya Adriansyah

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Adriansyah, Arya

Aligning Observed and Modeled Behavior/ by Arya Adriansyah.
Eindhoven: Technische Universiteit Eindhoven, 2014. Proefschrift.

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

ISBN: 978-90-386-3574-3

Keywords: Business Process Management, Conformance Checking, Process Mining, Alignments

SIKS Dissertation Series No. 2014-07
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The work in this thesis has been sponsored by NWO under the REPLAY project (612.063.919).

Printed by Gildeprint Drukkerijen (www.gildeprint.nl)

Cover design by Arya Adriansyah

Aligning Observed and Modeled Behavior

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 donderdag 3 april 2014 om 16.00 uur

door

Arya Adriansyah

geboren te Bandung, Indonesië

Dit proefschrift is goedgekeurd door de promotor:

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

Copromotor:
dr. B.F. van Dongen

Abstract

Aligning Observed and Modeled Behavior

The availability of process models and event logs is rapidly increasing as more and more business processes are supported by IT. On the one hand, most organizations make substantial efforts to document their processes, while on the other hand, these processes leave footprints in their information systems. Although it is possible to extract event logs from today's systems, the relation between event logs and process models is often identified using heuristics that may yield misleading insights. In this thesis, techniques to *align* event logs and process models are explored. Based on the obtained alignments, various analysis techniques are developed. The techniques are evaluated against both artificial and real-life process models and event logs.

A memory-efficient technique to compute alignments between event logs and process models has been developed. Given an event log and a process model, low-level deviations, i.e., observed activities that are not allowed according to the model and the other way around, are explicitly identified. The technique can also be used to identify high-level deviations such as swapped and replaced activities.

Our technique is applied to problems occurring in different domains. Unlike earlier approaches, alignment-based conformance checking techniques are shown to be robust against peculiarities of process models, such as duplicate and invisible tasks. Alignment-based conformance metrics, such as fitness and precision, are shown to be more intuitive and can deal with multiple level of noise in event logs. Various visualizations of alignments provide powerful diagnostics to identify the context of frequently occurring deviations between process executions and prescribed process models. Applying data mining techniques to alignments yields root causes of deviations between the observed behavior in an event log and the modeled behavior in a process model. Alignments also improve the robustness of performance measurements based on event logs and process models, even if the logs are deviating from the models.

From a computational point of view, computing alignments is extremely expensive. However, the obtained results indicate that alignments not only provide a theoretically solid basis for analysis based on both models and process executions, but are also able to handle problems of real-life complexity.

Contents

Part I

Introduction

Chapter 1

Overview

The increased level of competition between organizations forces individual organizations to perform in the best way possible. Many approaches to improve performance of organizations such as Six Sigma [123], Total Quality Management [132], and Business Process Re-engineering [73] show that efficient business processes are one of the keys to improve their overall performance. Therefore, it is no surprise that Business Process Management (BPM) has become one of the top concerns for many organizations nowadays. Consulting firms such as Gartner even put business process as one of the top 5 concerns for organizations for the last 7 years consecutively [113, 126–128, 159].

A *business process* is a set of activities that are performed and need to be coordinated in an organizational and technical environment to realize a business goal [205]. A business process can be as simple as two activities performed in a sequence, or it can be as complex and involve hundreds of activities to be performed in parallel with possible synchronization and loops. Regardless of their size, documentation of business processes – particularly in form of *process models* – is important in many organizations. Such documentation provides an effective and efficient way to get insights into business processes. Process models are also often useful for analysis purposes. Moreover, recent laws and regulations such as Sarbanes-Oxley may enforce organizations to document their processes.

The rapidly changing business environment forces people and organizations to be highly flexible and allow deviations from documented process models. For example, a patient handling process in a hospital normally starts with the registration of a patient followed by a general examination before any further action is performed. However, in case of emergency, a patient of the hospital may go directly to operation table and skip both registration and general examination. Hence, the execution of the process may not always conform to its model. Deviations from a prescribed process model influence the correctness of all analysis based on the models. Therefore, it is important to *align* the observed behavior occurring in reality to the ones described in process models.

Many organizations nowadays use information systems to support their business process. Such systems typically log all events that occurred during process executions, i.e., they record all *observed behavior*. This information can be exploited to identify deviations to documented process models and further extended to provide other types of analysis.

In this thesis, we focus on *the analysis based on alignments between the observed behavior in event logs and the modeled behavior in process models*. Alignments provide insights into deviations that can be exploited further for deviation analysis. In this chapter, we

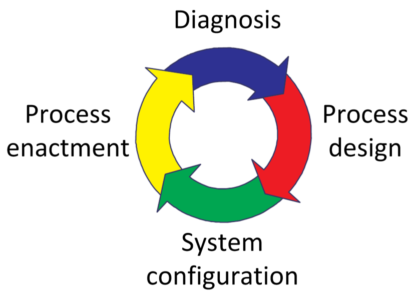


Figure 1.1: Business Process Management Lifecycle [185, 205].

describe the context of our research. First, we provide an overview of Business Process Management (BPM) and the advancement of data-oriented analysis approaches in Section 1.1 and Section 1.2. This provides the necessary background of our research. Section 1.3 explains the Process Mining research area and positions this thesis within the area. The research challenges addressed in this thesis are explained briefly in Section 1.4. Section 1.5 highlights the contributions and describes the structure of this thesis.

1.1 Business Process Management

The idea to support business processes emerged in late 1970s. Zisman [210] investigated an office automation approach to improve performance of business processes. However, the absence of computer networks made it impossible to develop these ideas any further at the time they were presented. Only in late 90s, the concepts could be fully implemented in form of Workflow Management Systems (WFMS) [186]. Workflow supports the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [96]. WFM emphasizes the use of software to support the execution of operational processes [185]. Workflow Management Systems (WFMS) [96] are the software products to realize such support.

Experience shows that a WFMS alone is not sufficient to support organizations during the whole life-cycle of business processes. In particular, WFM does not support diagnosis of executed business process. Thus, *Business Process Management* (BPM) emerged as an extension of WFM. BPM covers all areas covered by WFM and some other important areas that were not (fully) covered by WFM. While WFM emphasizes mostly on configuration phase, BPM covers all phases of business process life-cycle as shown in Figure 1.1. BPM supports business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information [185]. In the *design* phase, the processes are (re-)designed and analyzed. Designs are implemented in the *configuration* phase by configuring an information system that supports business process. Such a system is called a Process-Aware Information System [60]. The *enactment* phase starts when business process are executed using the configured systems. Finally, operational processes are analyzed in the *diagnosis* phase to identify problems and possible improvements.

Business Process Analysis (BPA) techniques are covered by BPM but not by WFM. BPA focuses on the diagnosis phase of BPM life-cycle. *Business Activity Monitoring (BAM)* is

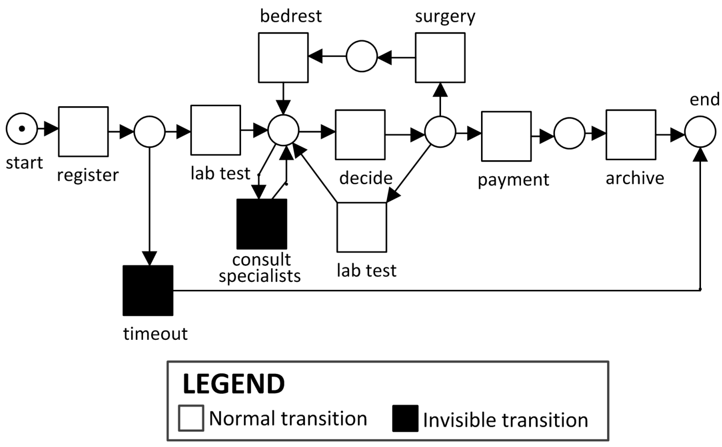


Figure 1.2: Petri net model for handling patients in a hospital.

an emerging area in BPA that uses data logged by information systems to diagnose operational processes. Process model often plays a crucial role in BPA. Process models show how activities in the process must be performed. They provide insights into business processes and means of analysis. Moreover, process models can be used to enact process executions [83].

A process model can be formal (i.e., having clear definition and semantics) or informal. Formal models are unambiguous and often supported by analysis techniques and tools [167]. However, such models can be overly complex as all details need to be modeled explicitly. In contrast, informal models may ignore some details and hence are perceived to be simpler. However, this also means that such models can be ambiguous and even yield misleading insights. For analysis purposes, models expressed in terms of a formal process modeling language are often more valuable than informal ones. In addition, graphical models are often preferred over ones without as they are easier to be communicated.

Petri nets [119] are an example of a formal process modeling language supported by many analysis techniques while providing a clear graphical notation. Figure 1.2 shows an example of a patient handling process in a hospital, modeled in terms of a Petri net. Rectangular nodes are called *transitions*, while circular nodes are called *places*. Arcs show dependencies between transitions. *Tokens* reside in places. The distribution of tokens in places defines the state of the process. A transition in a Petri net is typically labeled over an activity. However, a transition in a Petri net may also have been introduced for routing purposes only. In a Petri net, a transition can be executed, i.e., *fired*, if all input places of the transition have tokens.

Figure 1.2 shows that a patient must first register (register) and then make an appointment for a lab test (lab test). Then, a doctor decides (decide) whether the patient needs to have another test, go for a surgery (surgery), or go home immediately. A patient who undergoes a surgery must stay temporarily in the hospital (bedrest) until the doctor decides what to do next. Before the patient goes home, he needs to pay his bill (pay). The process ends when an administration officer of the hospital archives the whole treatment (archive). If a registered patient does not show up during a scheduled lab test, the process for the patient terminates automatically after a certain time has passed. Transition labeled timeout in Figure 1.2 is a routing transition that explicitly models the automated termination. Occurrences of routing transitions are typically not observed by information systems, thus they are *invisible*. We use black-colored transitions to mark such invisible transitions in a Petri net. Besides modeling the possible routings of a process, invisible transitions are also used to model activities that may change the state of processes but not directly observable from information systems. For example in Figure 1.2, a doctor may consult some specialists (consult specialists) as many times as needed before making any decisions. Activity consult specialists is typically performed without any support from the hospital's information systems, thus it is not observable and hence modeled as an invisible transition.

A formal process model such as the one shown in Figure 1.2 allows for many types of analysis in various phases of BPM lifecycle. In the *design* phase, undesired properties such as deadlocks and live-locks can be identified from the model. Simulations can be performed to determine the expected time needed to finish an instance of the process. Such analysis can be used to ensure that a certain level of quality is met before the configuration phase is started. In the enactment phase, some recommendations regarding the selection of the transitions to fire next can be given based on both the model and historical information. In the diagnosis phase, comparison between recorded process executions with the model may reveal deviations and frequently executed activities.

1.2 Data Analysis

The rapid evolution of digital technology in the past three decades has brought us into the so-called "big data" era. For example, Obama's government recently announced that big data in size of terabytes (10^{12} bytes) is generated daily from experiments in the office of Basic of Energy Sciences. E-bay, the online auction and web shopping company, processes around 50 petabytes daily – or 50 quadrillion bytes [92]. Facebook, the popular social media company, processes more than 500 terabytes of data daily [163]. Torrents of digital data are generated daily and will continue to grow exponentially for the foreseeable future [103]. Furthermore, they are available in various forms. Transactions, logs, click streams, sensor data, audio, video, and texts are some types of data that are now available in massive quantities.

Information systems nowadays often record events that occurred during business process executions, thus providing an abundance of historical data on how processes were performed. A study reported in [158] shows that the top three sources of big data are transactions, log data, and events. All of them are often generated as a by-product of process executions. The same report also shows that data mining, visualization, predictive modeling, and optimization are the four most frequently used techniques to analyze big data after querying and report. Note that all techniques can be related to process analysis, but none of them takes the process model explicitly into account. Data mining is a field of study that covers various approaches to extract knowledge from large amounts of data [78]. It involves techniques such as machine learning and data visualization. Predictive modeling covers various approaches based on statistics to predict future behavior based on historical data, while optimization techniques cover various ways to obtain a mathematical model in order to identify variable values that maximize/minimize target formula.

Thus, there is an abundance of process-related data that offers a wealth of information for todays organizations, but it is rarely exploited to provide meaningful insights on business processes. As a consequence, the analysis results may be incomprehensive. In

Table 1.1: Example of an event log

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Transaction Type	...
1	1023	20-10-2013 11:50	register	John	complete	...
	1024	22-10-2013 08:10	lab test	Tifania	complete	...
	1025	22-10-2013 10:04	decide	Fitriani	complete	...
	1026	22-10-2013 10:20	payment	Arya	complete	...
	1027	23-10-2013 08:05	archive	Kate	complete	...
2	1028	20-10-2013 12:15	register	John	complete	...
	1029	22-10-2013 09:10	lab test	Tifania	complete	...
	1030	22-10-2013 10:00	decide	Fitriani	complete	...
	1031	25-10-2013 08:00	surgery	Jim	complete	...
	1032	25-10-2013 08:45	bedrest	Kate	complete	...
	1033	25-10-2013 09:10	decide	Fitriani	complete	...
	1034	25-10-2013 10:10	payment	Arya	complete	...
	1035	25-10-2013 12:10	archive	Kate	complete	...
3	1036	20-10-2013 13:30	register	John	complete	...
	1037	20-10-2013 13:40	surgery	Tifania	complete	...
	1038	20-10-2013 14:40	bedrest	Johann	complete	...
	1039	20-10-2013 15:30	decide	Fitriani	complete	...
	1040	23-10-2013 08:00	lab test	Tifania	complete	...
	1041	23-10-2013 09:30	payment	Arya	complete	...
	1042	23-10-2013 10:00	archive	Kate	complete	...
4	1043	20-10-2013 10:50	register	John	complete	...
...	...	...	...	...	...	...

business processes, activities are related one after another. A non-optimal execution of an activity in a process may influence the execution of other activities within the same process. Suppose that an activity in a process takes much longer than other activities in the same process, i.e., the activity is a bottleneck in the process. The causes of such a bottleneck may not be directly related to the activity, but can also be related to other activities that are executed before it. Without considering the process behind the activity, the root cause of such a bottleneck may not be correctly identified.

1.3 Process Mining

Process mining bridges the gap between the process-oriented nature of BPM and the data-oriented nature of machine learning/data mining. The starting point of process mining is the *observed* behavior of process executions, stored in so-called *event logs*. Table 1.1 shows an example of an event log of the patient handling process shown in Figure 1.2. Note that all events are already grouped per case. In this example, the completion of each activity is recorded as an event in the log. The additional information related to the execution such as the resource, timestamp, and case ID of the event may also be recorded as shown in Table 1.1.

Process mining is a research discipline that discovers, monitors, and improves real processes (not the assumed processes) by extracting knowledge from event logs readily available from today's systems [171]. Figure 1.3 shows an overview of the research area covered by process mining. As shown, process mining links the modeled behavior on one hand and the observed behavior on the other hand. There are three types of process mining techniques: *discovery*, *conformance*, and *enhancement*.

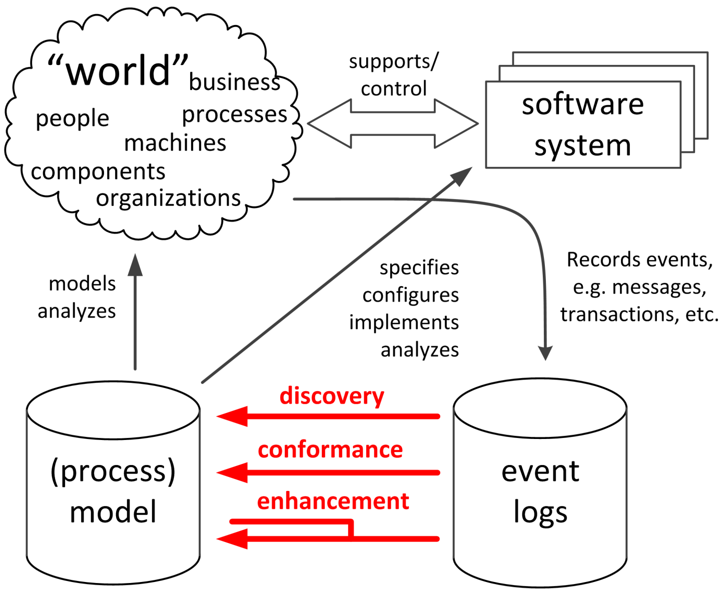


Figure 1.3: Three types of process mining techniques: (1) Discovery, (2) Conformance, and (3) Enhancement [171].

- A process *discovery* technique takes an event log as input and produces a model that best describes the behavior observed in the log. For example, the α algorithm [1] constructs a Petri net model from an event log. The goal of process discovery techniques is not limited to constructing models that show the control-flow of activities, but also other dimensions such as uncovering the social network between resources that perform activities [182]. Process discovery techniques are mostly useful to provide *insights into what occurs in reality*.

- *Conformance checking* techniques take a process model and an event log of the same process as input. Conformance checking compares the observed behavior in the log with the behavior allowed by the model. If the model allows for more behavior than the behavior observed in the log or vice versa, the log is said to be not conforming to the model. An example of a conformance checking technique is the token-based replay approach described in [151]. Conformance checking techniques are mostly useful in situations where process models do not strictly enforce process executions, i.e., deviations to prescribed process model may occur. Such techniques can be used to *identify where and when deviations occur, and measure the severity of such deviations*.

- An *enhancement* technique takes both an event log and a process model to *extend or improve the model with information extracted from the log*. There are various types of enhancement that one can perform, such as *repairing* a process model to better reflect reality. Given a Petri net and an event log, the model repair approach described in [62] adds extra transitions to the original net to better reflect the observed behavior in the log. Another type of enhancement is the *extension* of process models with information extracted from event logs. An example of such an enhancement is the approach to derive a simulation model, given a process model and an event log of the same process [149]. For example, durations, allocations, rules, and routing probabilities are learned from the event log.

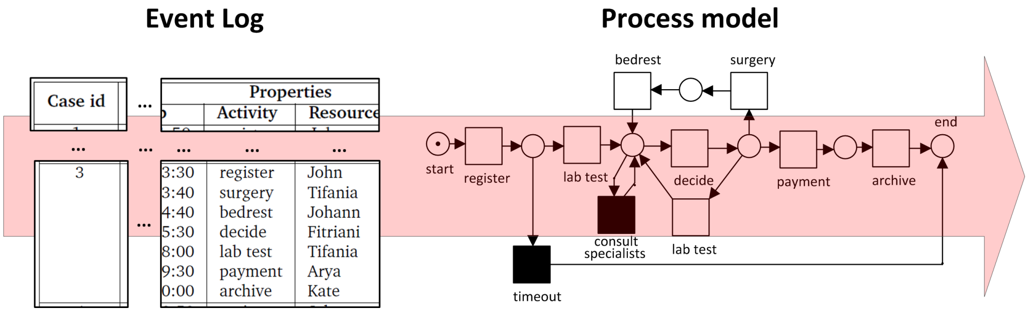


Figure 1.4: Replay maps logged events to transitions in Petri nets. The log and net are taken from Table 1.1 and Figure 1.2 respectively.

1.4 Challenges in Conformance and Enhancement

In this thesis, we focus on two types of process mining: *conformance* and *enhancement*. Hence, we assume that a process model and an event log are given. The model may have been discovered through process discovery or made by hand. One of the main challenges to perform both conformance checking and enhancement is to find the best way in which observed behavior in event logs can be *replayed* on process models. Suppose that we consider process models in the form of Petri nets. Replay takes an event log and a Petri net and maps occurrences of events in the log to transition executions in the net. In a situation where the net is relatively simple and only allows for the behavior observed in the event log and vice versa, mapping events to transitions is trivial. Problems arise when the observed behavior in the log is not following the same behavior as the behavior allowed by the net. Take for example a fragment of an event log and a net in Figure 1.4. The execution of the first event with label register in the log is allowed according to the net, but the net does not allow for the execution of transition labeled surgery directly after the transition labeled register. A question that one may naturally ask in such a situation is *how to continue replay if the events are not allowed according to the model*.

Another challenge of replay is to *deal with peculiarities of process models*. A Petri net may have multiple transitions with the same label, i.e., *duplicate transitions*. Figure 1.4 is an example of such a net. There are two transitions in the net labeled lab test, i.e., two transitions have the same label. For any event labeled lab test in the log, it is not always trivial to determine which transitions it should be mapped to. Other problems may arise due to invisible transitions. Replay should also be able to identify the occurrence of invisible transitions based on observed behavior in the log. Take for example the net in Figure 1.4. Suppose that a patient made a registration but he didn't show up for the scheduled lab test. According to the net, the transition labeled timeout can be fired and hence the patient handling terminates properly. A proper replay approach should be able to identify that a timeout occurred in such a case. Recall that invisible transitions are not observable, hence they are not recorded in event logs. Nevertheless, it is often possible to infer their presence.

With the availability of large and complex processes, enabling replay for such processes also becomes a challenge from a performance point of view. More and more process models are constructed with complex control flow patterns. The reasonably cheap price for digital storage motivates many organizations to record torrents of data. Showing meaningful insights into executions of a process based on such a big data is non-trivial, especially in cases where deviations occur in many states of the process.

Given an event log and a process model, a replay technique is required to perform conformance checking but it is not sufficient to *measure the degree of deviations between the log and the model*. Such metrics are particularly useful to compare how good a model is in comparison with other models that also aim to describe the behavior recorded in the log. Genetic process discovery algorithms, e.g., [47], rely on conformance metrics to construct a "good" process model from a given event log. Such measurements are also useful to quantify the degree of confidence in all log-model-based analysis results. Furthermore, if deviations exist, another important challenge to tackle is to *provide reliable diagnostics describing deviations*. Diagnostic information on deviations, e.g., transitions that are skipped, patterns of reoccurring deviations, etc., provide insights into possible causes of deviations. With such insights, either some actions can be taken to prevent further deviations or models can be repaired to better reflect reality. Auditors, security analysts, and business process analysts are some parties that can gain the most benefits from such insights.

Performance measurement is a crucial aspect for many organizations. In organizations where BAM applications are used to monitor process executions and executions strictly follow predefined process models, performance values can be measured trivially and projected onto the models to identify bottlenecks in the overall process. However, in systems where deviations occur, measuring performance is far from trivial. Some activities may be skipped, thus events may occur in a different order than the ones allowed by prescribed process models. Measuring performance based on the order of events and projecting the result to process models may yield misleading results. Therefore, the main challenge in measuring performance is *how to deal with deviations*.

1.5 Contribution and Thesis Structure

The contributions in this thesis can be summarized as follows.

- A *robust approach to replay event logs on process models*. Given an event log and a Petri net, we show in this thesis that it is possible to *align* observed behavior in the log to the net such that the alignment provides a robust mapping between events in the log and transitions in the model even in the presence of invisible and duplicate transitions in the net. Furthermore, the approach shows explicitly where, when, and why deviations occur, thus providing a basis for further analysis. A *memory-efficient approach* to compute alignment has been invented to enable alignment-based analysis for complex and large-sized logs and models. The approach is extendable to also show high-level deviations, e.g., replaced transitions and swapped activities.
- A set of *conformance metrics* based on alignments. Given an event log and a Petri net, these metrics yield intuitive insights into the conformance between the log and the net even if the log is non fitting.
- A set of *visualization techniques to diagnose root cause of deviations*. Each visualization shows different insights regarding deviations (e.g., frequently occurring deviations, context of deviations, or detailed information on deviation per case). They are complementary and altogether offer powerful analysis tools to diagnose the root causes of deviations.
- A *robust performance measurement* approach based on alignments. Given an event log and a Petri net, we show that performance can be measured accurately from

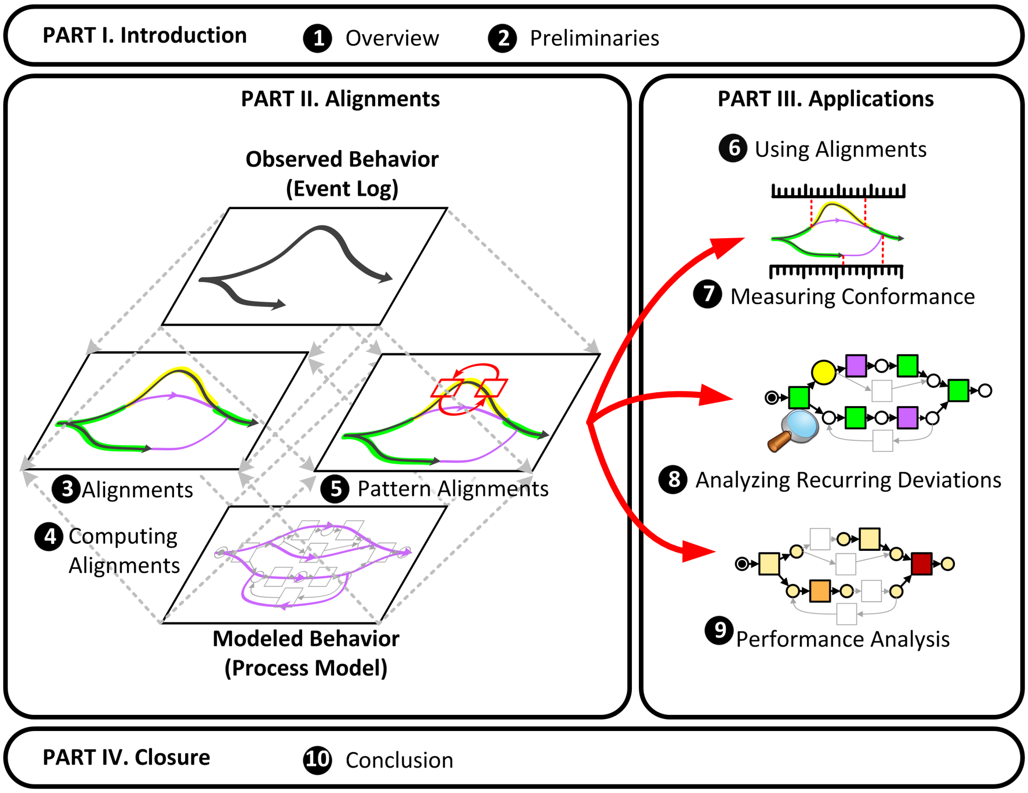


Figure 1.5: Thesis structure.

them even if the log is not perfectly fitting.

All of the introduced techniques have been designed and implemented in ProM 6, an open source process mining framework[1].

Figure 1.5 shows the structure of this thesis. It is divided into four main parts. The first part, Part I, provides an introduction to this thesis. It consists of an introduction (Chapter 1) and preliminaries (Chapter 2).

Part II explain alignment-related concepts and their computation. Chapter 3 discusses existing approaches to compare observed and aligned behavior and motivates why an approach based on alignments was chosen. Furthermore, it provides a formalization of the notion of alignments. Efficient approaches to compute alignments are explained in Chapter 4. Chapter 5 presents possible extensions to the basic alignment concepts. In particular, deviations at higher levels of granularity (i.e., pattern-based deviations) are investigated.

Part III describes various applications of alignments. Chapter 6 explains how alignments are formalized and how they are constructed using so-called "oracle" functions. In Chapter 7, we describe approaches to measure conformance between event logs and process models using alignments. Chapter 8 explains alignment-based visualizations that can be used to provide insights into deviations. An approach to robustly measure performance based on event logs and process models is given in Chapter 9.

[1]see `http://www.processmining.org`

Part IV concludes the thesis. Chapter 10 summarizes the main results and discusses possible extensions of the work presented in this thesis.

Chapter 2

Preliminaries

This chapter introduces the notations that are used in the remainder of this thesis. Basic notations are introduced in Section 2.1. Section 2.2 introduces various notations for graphs. The event log notion is formalized in Section 2.3. An overview of Petri nets and related concepts is provided in Section 2.4, while other process modeling formalisms are explained in Section 2.5.

2.1 Basic Notations

In this section, we introduce the basic notations for sets, functions, matrices, vectors, sequences, multisets, and tuples.

Definition 2.1.1 (Sets and Functions)
A *set* is a possibly infinite collection of *elements*. We denote a finite set by listing its elements between braces, e.g., a set A with elements a, b and c is denoted as $\{a, b, c\}$. The *empty set*, i.e., the set with no elements, is denoted by $\emptyset$. Let $A = \{a_1, \ldots, a_n\}$ be a *set* of size $n \in I\!N$. $|A| = n$ denotes the size of set A and $\mathcal{P}(A)$ is the powerset of set A, e.g., $\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$.

Let $A = \{a, b, c, d\}$ and $B = \{a, c, d, e\}$ be non-empty sets. The *union* of A and B, denoted $A \cup B$ is the set containing all elements of either A or B, e.g., $A \cup B = \{a, b, c, d, e\}$. The *intersection* if A and B, denoted $A \cap B$, is the set containing elements of both A and B, e.g., $A \cap B = \{a, c, d\}$. The *difference* between A and B, denoted $A \setminus B$ is the set containing all elements of A that does not exists in B, e.g., $A \setminus B = \{b\}$.

Let A and B be non-empty sets. A *function* f from A to B, denoted $f : A \to B$, is a relation from A to B, where every element of A is associated to an element of B. A *partial function* g is a relation from A to B, denoted $g : A \not\to B$, where some elements of A is associated to elements of B, i.e., g may be undefined for some elements of A. For all (partial) functions f, Dom(f) and Rng(f) denote the domain and range of function f respectively.

Function f is *surjective* if for all $b \in B$, there exists $a \in A$ such that $f(a) = b$. f is an *injective* function if for all $a, a' \in A : f(a) = f(a')$ implies $a = a'$. Function f is *bijective* if it is both surjective and injective.

We assume all sets to be totally ordered, i.e., for any set $A = \{a_1, \ldots, a_{|A|}\}$, we assume a bijection $\lambda : A \to \{1, \ldots, |A|\}$ exists, and for all $1 \leq i \leq |A|$, we write $A[i]$ as a shorthand for $\lambda^{-1}(i)$. $idxOf(a_i, A)$ denotes the ordering of $a_i \in A$ in set A, such that for all $A[i], idxOf(A[i], A) = i$. The ordering of elements in a set is respected by all of its subsets, i.e., for all $A' \subseteq A$, for all $a_i, a_j \in A'$ where $idxOf(a_i, A) < idxOf(a_j, A)$: $idxOf(a_i, A') < idxOf(a_j, A')$.

┘

In the remainder of this thesis, we typically use uppercase letters to denote sets and lowercase letters to denote the elements of that set. $I\!N$ is the set of natural number, i.e., $I\!N = \{0, 1, \ldots\}$, $I\!R$ is the set of all non-negative real values, and $I\!R^+$ is the set of all non-negative real values without 0, i.e., $I\!R^+ = I\!R \setminus \{0\}$.

Definition 2.1.2 (Matrix and Vector)
A *matrix* is a square of array of numbers. Let [M] be a matrix of size $m \times n$. $[\mathbf{M}]_{i,j}$ denotes the element of matrix [M] in the i-th row and j-th column where $1 \leq i \leq m$ and $1 \leq j \leq n$. $[\mathbf{M}]^T$ of size $n \times m$ is the *transpose* of [M] such that for all $1 \leq i \leq m, 1 \leq j \leq n : [\mathbf{M}]_{i,j} = [\mathbf{M}]^T_{j,i}$.

Let $[\mathbf{M_1}]$ and $[\mathbf{M_2}]$ be a pair of matrices with the same size $m \times n$. The addition of two matrices $[\mathbf{M_1}]$ and $[\mathbf{M_2}]$, denoted $[\mathbf{M_1}] + [\mathbf{M_2}] = [\mathbf{M_3}]$ such that for all $1 \leq i \leq m, 1 \leq j \leq n, [\mathbf{M_3}]_{i,j} = [\mathbf{M_1}]_{i,j} + [\mathbf{M_2}]_{i,j}$. Substraction two matrices is defined in the same way as addition by replacing summation '+' with substraction '-'. The *dot product* of $[\mathbf{M_1}]$ and $[\mathbf{M_2}]$, denoted $[\mathbf{M_1}] \cdot [\mathbf{M_2}] = \sum_{1 \leq i \leq m, 1 \leq j \leq n} [\mathbf{M_1}]_{i,j} \cdot [\mathbf{M_2}]_{i,j}$.

Let $[\mathbf{M_4}]$ be a matrix of size $n \times o$. The *cross product* of $[\mathbf{M_1}]$ and $[\mathbf{M_4}]$ is a matrix $[\mathbf{M_5}]$ of size $m \times o$, denoted $[\mathbf{M_1}] \times [\mathbf{M_4}] = [\mathbf{M_5}]$, such that for all $1 \leq i \leq m, 1 \leq j \leq o, [\mathbf{M_5}]_{i,j} = \sum_{r=1}^{n} [\mathbf{M_1}]_{i,r} \cdot [\mathbf{M_4}]_{r,j}$.

A *row vector* $\vec{v}$ of size m is a matrix of size $1 \times m$. Similarly, a *column vector* $\vec{w}$ of size n is a matrix of size $n \times 1$. $\vec{v}_i$ and $\vec{w}_i$ refer to the value of the i-th column in $\vec{v}$ and the value of the i-th row in $\vec{w}$ respectively. ┘

Definition 2.1.3 (Sequences)
Let A be a set. A^* denotes the set of all finite *sequences* over A. $\langle\rangle$ denotes an empty sequence. A sequence $\sigma = \langle \sigma[1], \ldots \sigma[n] \rangle$ can be represented by listing its elements between angled brackets, where $\sigma[i]$ refers to the i-th element of a sequence and $|\sigma| = n$ denotes the length of σ. For all $a \in A, \sigma(a)$ counts the number of occurrences of a in σ, i.e. $\sigma(a) = |\{1 \leq i \leq |\sigma| \mid \sigma[i] = a\}|$.

Concatenation of two sequences σ and σ' is denoted with $\sigma \cdot \sigma'$. Similarly, concatenation of an element $a \in A$ and a sequence σ is denoted $a \cdot \sigma$. Prefix sequences are denoted with $<$, such that $\sigma < \sigma'$ if and only if there is a sequence $\sigma'' \neq \langle\rangle$ with $\sigma' = \sigma \cdot \sigma''$.

When we iterate over $a \in \sigma$, we refer to each unique element in the sequence σ, e.g., for all $f : A \to I\!N$, $\sum_{a \in \sigma} f(a) = \sum_{1 \leq i \leq |\sigma|} f(\sigma[i])$. For all $A' \subseteq A$, $\sigma_{\downarrow A'}$ denotes the projection of a sequence $\sigma \in A^*$ on A', e.g., $\langle a, a, b, c \rangle_{\downarrow \{a,c\}} = \langle a, a, c \rangle$. The *parikh vector* $\vec{\sigma}$ of a sequence σ over A is a column vector, such that $\vec{\sigma} = (\sigma(A[1]), \ldots, \sigma(A[|A|]))^T$, i.e., $\forall_{1 \leq i \leq |A|}\ \vec{\sigma}_i = \sigma(A[i])$. The sequence of set A, denoted $seq(A)$, contains all elements of A based on their ordering, i.e., $seq(A) \in A^*, |seq(A)| = |A|$ and for all $1 \leq i \leq |A|, seq(A)[i] = A[i]$. For all $\sigma \in A^*$ and $1 \leq i \leq j \leq |\sigma| : \sigma_{[i..j]}$ denotes the subsequence of σ from index i to j, i.e., $\sigma_{[i..j]} = \langle \sigma[i], \sigma[i+1] \ldots \sigma[j] \rangle$. $rv(\sigma)$ denotes the reverse of sequence σ, i.e., $rv(\sigma) = \langle \sigma[|\sigma|], \ldots, \sigma[1] \rangle$.

┘

Definition 2.1.4 (Multi-sets(bags))
Let A be a set. A *multi-set* m over A is a function $m : A \to I\!N$. $\mathbb{B}(A)$ denotes the set of all multi-sets over a finite domain A. We write e.g., $m = [a, b^2]$ for a multi-set m over A

where $a, b \in A, m(a) = 1, m(b) = 2$, and $m(c) = 0$ for all $c \in A \setminus \{a, b\}$. Furthermore, a set $S \subseteq A$ can be viewed as a bag where each element occurs once, i.e., $m : S \to \{1\}$.

Let $m_1 \in \mathbb{B}(A)$ and $m_2 \in \mathbb{B}(A)$ be two multi-sets. We denote the union of two multi-sets $m_3 = m_1 \uplus m_2$, i.e., $m_3 \in \mathbb{B}(A)$ where for all $a \in A : m_3(a) = m_1(a) + m_2(a)$. The difference between two multi-sets is denoted $m_3 = m_1 - m_2$ such that for all $a \in A : m_3(a) = \max(0, m_1(a) - m_2(a))$. The presence of an element in a multi-set $(a \in m_1) \Leftrightarrow (m(a) > 0)$, the notion of submulti-sets $(m_2 \leq m_1) \Leftrightarrow \forall_{a \in A}\ m_2(a) \leq m_1(a)$, and the size of multi-sets $|m_1|$ are defined in a straightforward way, e.g., $|[a, b^2, c^5]| = 8$. When enumerating elements of a multi-set, we do it for each element uniquely, e.g., $\sum_{n \in [a^2, b^3, c^2]} n = 2a + 3b + 2c$. For all set $S \subseteq A$, $[a \in S]$ denotes a multi-set that contains all elements of S precisely one, e.g., $[x \in \{a, b, c\}] = [a, b, c]$. Similarly, for all sequences $\sigma \in A^*$, $[a \in \sigma]$ denotes a multi-set of all elements in σ, e.g., $[x \in \langle a, b, b, c, d, a\rangle] = [a^2, b^2, c, d]$.

The *parikh vector* $\vec{m}$ of m is a column vector, such that $\vec{m} = (m(A[1]), \ldots, m(A[|A|]))^T$, i.e., $\forall_{1 \leq i \leq |A|}\ \vec{m}_i = m(A[i])$. For all $A' \subseteq A$, $m_{\downarrow A'} \in \mathbb{B}(A)$ denotes the projection of m to domain A', such that for all $a' \in A'$, $m_{\downarrow A'}(a') = m(a')$ and $m_{\downarrow A'}(b') = 0$ for all $b' \in A \setminus A'$. ⌟

Definition 2.1.5 (Tuple)
Let A be a set and let $t = (a_1, a_2, \ldots, a_n) \in A \times \ldots \times A$ be a tuple of n elements. $\pi_i(t)$ refers to the i-th element of tuple t, e.g., Let $(a, b) \in A \times A$ be a tuple of 2 elements (i.e., *pair*), $\pi_1((a, b)) = a$ and $\pi_2((a, b)) = b$. We generalize this notation to sequences of tuples, e.g., for all $\sigma \in (A \times A)^*$, $\pi_i(\sigma) = \langle \pi_i(\sigma[1]), \ldots, \pi_i(\sigma[|\sigma|])\rangle$. ⌟

2.2 Graphs

Process models are represented in terms of graphs and have a corresponding graphical representation. A graph consists of nodes and arcs that connect them. A directed graph is a graph whose edges have directions. In this thesis, we consider graphs whose arcs have both directions and labels. Such graphs are called *labeled directed graphs*. We formalize labeled directed graphs as follows.

Definition 2.2.1 (Labeled Directed Graphs)
A *labeled directed graph* is a tuple $DG = (NG, EG, LG)$ where NG is a set of nodes, LG is a set of labels, and $EG \subseteq NG \times LG \times NG$ is a set of labeled edges. ⌟

Many real-life problems are solved by modeling them as graph-related problems and then applying graph-related techniques. For example, the technique to find a shortest path between two nodes in a graph and find a shortest path to visit all nodes in a graph underlies many navigation systems, distribution planning, and network design we have nowadays. One of the basic concepts in graph theory is the connection between nodes, i.e., *path*, which is formalized as follows.

Definition 2.2.2 (Path)
Let $DG = (NG, EG, LG)$ be a labeled directed graph. For all nodes $n, n' \in NG$, a *path* from n to n' is a sequence of edges $\sigma \in EG^*$, where $\sigma = \langle\rangle \implies n = n'$ and $\sigma \neq \langle\rangle \implies \pi_1(\sigma[1]) = n, \pi_3(\sigma[|\sigma|]) = n'$, and for all $1 \leq i < |\sigma| : \pi_3(\sigma[i]) = \pi_1(\sigma[i+1])$. $\Psi_{DG}(n, n')$ is the set of all paths from n to n' in DG. The annotation DG can be omitted if the context is clear. ⌟

Most path-related problems require a distance notion between nodes, e.g., finding a path that connect two nodes with the shortest distance. We formalize the distance of path and shortest path as follows.

Definition 2.2.3 (Distance of path, Shortest path)
Let $DG = (NG, EG, LG)$ be a labeled directed graph. Let $dist : EG \to I\!R$ be a distance function. The *distance* of a path $dist(\sigma)$ is the sum of distances of all edges on the path σ where we abuse the distance function notation $dist$, such that $dist(\sigma) = \sum_{1 \leq j \leq |\sigma|} dist(\sigma[j])$. A path $\sigma \in \Psi(n, n')$ is a *shortest path* from n to n' if for all $\sigma' \in \Psi(n, n')$, $dist(\sigma) \leq dist(\sigma')$. ⌟

Given a graph, a distance function, a source node, and a set of target nodes in the graph, there are various approaches to find a shortest path from the source node to a target node. One of the most efficient approach to compute such shortest distance is the $\mathbb{A}^\star$ algorithm [50, 81]. The algorithm works in a breadth-first search manner while utilizing an estimation function to prune paths that can not lead to solutions.

Algorithm 1 shows a pseudocode of the $\mathbb{A}^\star$ algorithm. Given a directed graph, a source node, a set of target nodes, and an estimation function, the $\mathbb{A}^\star$ works by visiting a node in the graph and explore its direct successors iteratively until the visited node is a target node. In the first iteration, it visits the source node of the graph, explores all direct successors of the node, and then put them in a priority queue (see line 1-2). In the consecutive iterations, the algorithm visits a candidate node in the queue that most likely reaches the target node with the shortest distance (see line 4), explores all of its direct successors, and puts them in the queue (see line 13 and 18). For each queued node, the algorithm keeps track of (1) the shortest distance to reach the node from the source node, and (2) its direct node predecessor in a shortest path from the source node to the node (see line 11-12, 16-17). In each iteration, if there are multiple "best" candidates then one of them is selected randomly. If the selected candidate is a target node, the iterations stop and the path to reach the target node is constructed by recursively iterating the stored predecessors of the target node (see line 5-7).

Any estimation function used by the $\mathbb{A}^\star$ algorithm needs to be both *admissible* and *consistent* [50, 81]. We formalize the notions as follows:

Definition 2.2.4 (Admissible, consistent, and permissible function)
Let $DG = (NG, EG, LG)$ be a labeled directed graph and let $dist : EG \to I\!R$ be a distance function. A function $h_{(DG,dist)} : NG \times \mathcal{P}(NG) \to I\!R$ is *admissible* for DG and $dist$ if and only if for all $n \in NG, N_t \subseteq NG$:

- $h_{(DG,dist)}(n, N_t) = +\infty$ if for all $n_t \in N_t : \Psi_{DG}(n, n_t) = \emptyset$, and
- $h_{(DG,dist)}(n, N_t) \leq dist(\sigma)$ for all $n_t \in N_t, \sigma \in \Psi_{DG}(n, n_t)$ otherwise.

$h_{(DG,dist)}$ is *consistent* if and only if for all $(n, l, n') \in EG, N_t \subseteq NG : h_{DG}(n, N_t) \leq dist((n, l, n')) + h_{DG}(n', N_t)$. Furthermore, $h_{(DG,dist)}$ is *permissible* for DG and $dist$ if it is both admissible and consistent.

In the remainder of this thesis, we omit the annotations DG and $dist$ from h if the context is clear. Furthermore, if there is only one target node (i.e., $|N_t| = 1$) and the context is clear, we omit the powerset of nodes in the signature of the function, i.e., we write permissible function $h' : NG \to I\!R$ instead of $h : NG \times \mathcal{P}(NG) \to I\!R$ such that for all $n \in NG : h'(n) = h(n, N_t)$ with $N_t = \{n_t\}, n_t \in NG$. ⌟

Algorithm 1: Pseudocode of the $\mathbb{A}^\star$ algorithm

```
Initialize priority queue pqueue with the source node;
visitedNodesSet $\leftarrow \emptyset$ ;
while pqueue is not empty do
    currNode $\leftarrow$ best candidate node in pqueue (node with the minimum total
    distance+underestimation to the nearest target node);
    if currNode $\in$ the set of all target nodes then
        recursively iterate the predecessors of currNode until the source node to
        obtain a shortest path;
        return the shortest path;
    else
        forall succNode $\in$ set of all successors of currNode do
            if succNode $\in$ visitedNodesSet then
                if (stored best distance to reach succNode) $>$ (current distance to
                reach succNode) then
                    replace the values of stored best predecessor and total distance
                    for succNode with the current ones;
                    add succNode to pqueue;
                end
            else
                visitedNodesSet $\leftarrow$ visitedNodesSet $\cup$ {succNode};
                store predecessor and total distance to reach succNode;
                add succNode to pqueue;
            end
        end
    end
end
```

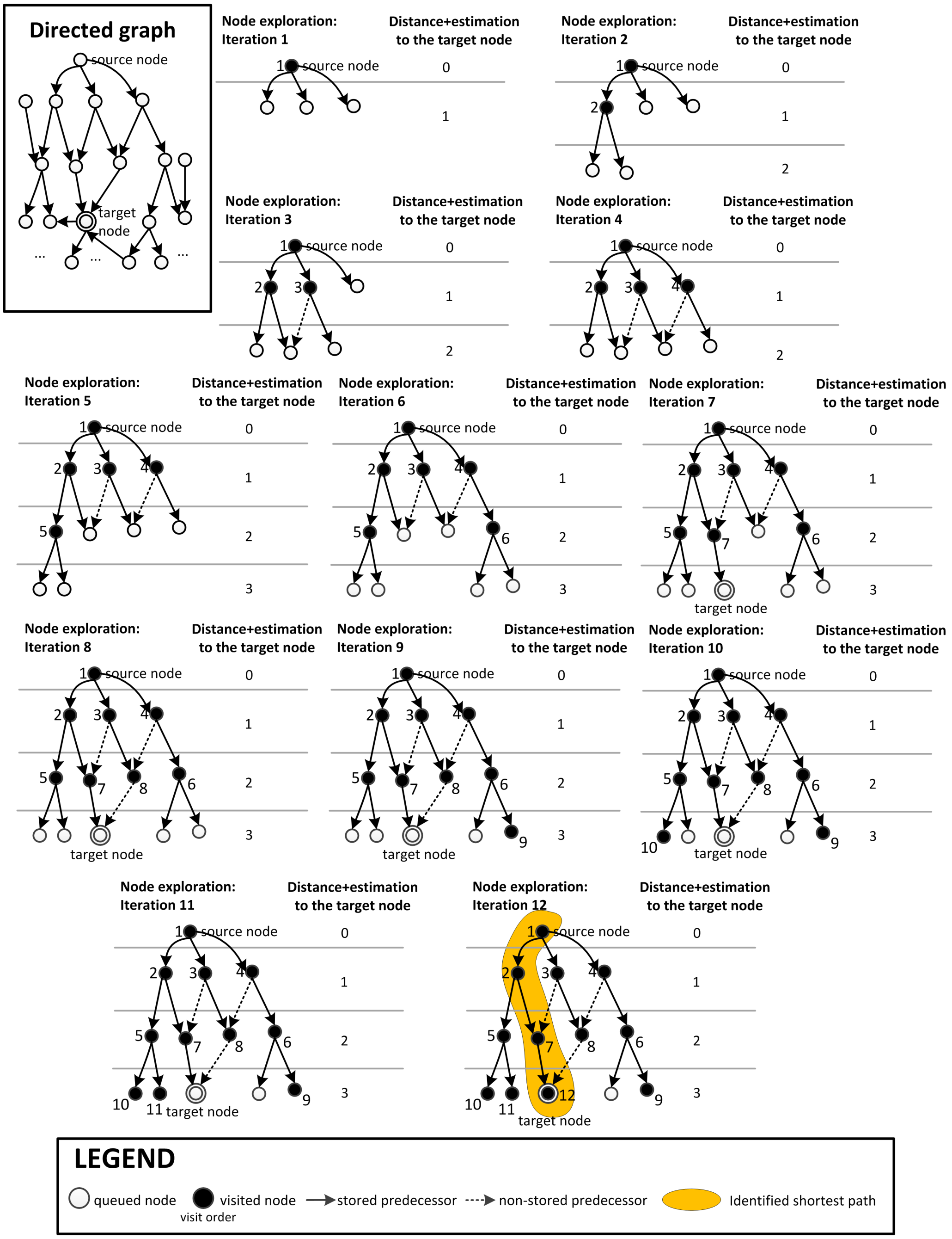

Directed graph
source node
target node
...
Node exploration: Iteration 1
Distance+estimation to the target node
0
1
2
3
Node exploration: Iteration 2
Node exploration: Iteration 3
Node exploration: Iteration 4
Node exploration: Iteration 5
Node exploration: Iteration 6
Node exploration: Iteration 7
Node exploration: Iteration 8
Node exploration: Iteration 9
Node exploration: Iteration 10
Node exploration: Iteration 11
Node exploration: Iteration 12
LEGEND
queued node
visited node
visit order
stored predecessor
non-stored predecessor
Identified shortest path

Figure 2.1: Top Left: a directed graph where all arcs have distance 1. The remainder of the figure shows how the $\mathbb{A}^*$ algorithm explores the nodes of the graph in each iteration to find a shortest path from the source node to the target node of the graph, using a permissible function that always returns 0 for all nodes.

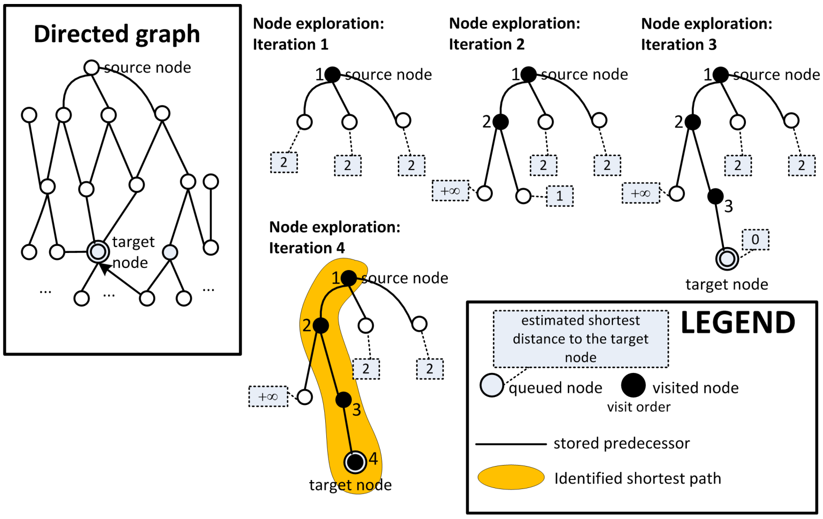

Directed graph
source node
target node
Node exploration: Iteration 1
Node exploration: Iteration 2
Node exploration: Iteration 3
Node exploration: Iteration 4
LEGEND
estimated shortest distance to the target node
queued node
visited node
visit order
stored predecessor
Identified shortest path

Figure 2.2: Top Left: the same directed graph where all arcs have distance 1 as shown in Figure 2.1. The remainder of the figure shows how the $\mathbb{A}^\star$ algorithm explores the nodes of the graph in each iteration to find a shortest path from the source node to the target node of the graph, using a permissible function that returns accurate distances to the target node (i.e., for any node in the graph, the function returns the shortest distance from the node to the target node or $+\infty$ if there is no path from the node to the target node). Fewer iterations are needed and fewer states need to be queued to find the same shortest path compared to the one shown in Figure 2.1.

It is easy to see that a function that always return 0 is a permissible function. Figure 2.1 illustrates how the $\mathbb{A}^\star$ algorithm works in a graph using a permissible function that always return 0. The number assigned to a node in the node exploration graph shows the visit ordering of the node. The highlighted path in the node exploration graph is the shortest path identified by the algorithm. Using such a permissible function, the algorithm works in a breadth-first search manner. All nodes with lower minimum total distance from the source node are investigated before other nodes with higher minimum total distances from the source node. If there are multiple candidate nodes in its priority queue with the same value of (minimum distance + underestimation to the nearest target node), one of them is chosen randomly. The algorithm stops when the selected candidate node is one of the target nodes. For example, after iteration 11 (see Figure 2.1) the priority queue contains two unvisited nodes. One of them is a target node. In iteration 12, the selected node is visited (i.e., it is chosen as the "best" candidate node) and therefore there is no need to visit the remainding nodes in the queue. Note that with a better permissible function, all nodes that are highly unlikely to be a part of a shortest path from the source node have higher underestimation values than other nodes and thus have less priority to be visited. Figure 2.2 shows how the $\mathbb{A}^\star$ algorithm explores the nodes in the same directed graph with a precise estimation function. With such a function, the algorithm requires fewer iterations to identify the same shortest path compared to the one shown in Figure 2.1. We refer to [50, 81] for further details of the $\mathbb{A}^\star$ algorithm.

Table 2.1: A fragment of some event log: each line corresponds to an event

Case id	Event id	Properties				
		Timestamp	**Activity**	**Resource**	**Transaction Type**	...
1	1023	20-10-2013 11:50	register	John	complete	...
	1024	22-10-2013 08:10	lab test	Tifania	complete	...
	1025	22-10-2013 10:04	decide	Fitriani	complete	...
	1026	22-10-2013 10:20	payment	Arya	complete	...
	1027	23-10-2013 08:05	archive	Kate	complete	...
2	1028	20-10-2013 12:15	register	John	complete	...
	1029	22-10-2013 09:10	lab test	Tifania	complete	...
	1030	22-10-2013 10:00	decide	Fitriani	complete	...
	1031	25-10-2013 08:00	surgery	Jim	complete	...
	1032	25-10-2013 08:45	bedrest	Kate	complete	...
	1033	25-10-2013 09:10	decide	Fitriani	complete	...
	1034	25-10-2013 10:10	payment	Arya	complete	...
	1035	25-10-2013 12:10	archive	Kate	complete	...
3	1036	20-10-2013 13:30	register	John	complete	...
...	...	...	...	...	...	...

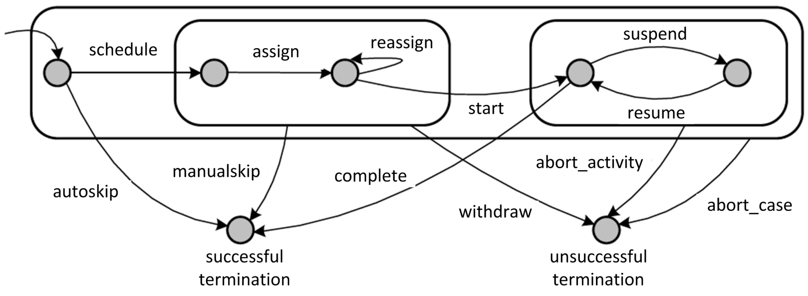


Figure 2.3: Standard transactional life-cycle model [171].

2.3 Event Logs

Table 2.1 shows a fragment of the log shown in Table 1.1. Recall that our example log stores some execution history of patient handling process in a hospital. An event log contains data related to a *single process*. For example, all events in Table 2.1 can be related to the same patient handling process. Each event in the log refers to a single *process instance*, often referred to as the *case*. Furthermore, an event is related to an execution of some *activity*. Other than case and activity, an event may have several other attributes, such as a *timestamps* or a *resource* (i.e., the person that executes the event).

To explicitly capture all information that may exist in event logs, we formalize complex events and their attributes as follows:

Definition 2.3.1 (Complex event, attribute)
Let $\mathcal{E}$ be the *complex event universe*, i.e., the set of all possible complex event identifiers. A complex event may have various *attributes*. Let $\mathcal{N}$ be the attribute universe. For all events $e \in \mathcal{E}$ and name $n \in \mathcal{N}$: $\#_n(e)$ is the value of attribute n for event e. ⌟

Often, activities may take time and have some life-cycle [171]. Figure 2.3 shows the standard life-cycle of activities [171]. Events are recorded when an activity instance changes state from one life-cycle to another. For example, events may be recorded at the moment activity instances are started or completed. Consider the following scenario where the standard activity life-cycle is used and events are recorded each time a transaction life-cycle changes. Suppose that an activity is *scheduled*, and then *assigned* to a resource. The resource *starts* the activity and then *completes* it. In this scenario, four events are recorded with the same activity attribute, but with different life-cycle transaction type (i.e., schedule, assign, start, and complete).

In the remainder of this thesis, let $\mathcal{C}$ be the case universe, let $\mathcal{A}$ be *the universe of all activities*, let $\mathcal{T}$ be the time universe, and let TY be the transaction type universe. We assume that all events $e \in \mathcal{E}$ have at least the following standard attributes:

- $\#_{case}(e) \in \mathcal{C}$ is the *case* associated to event e,
- $\#_{act}(e) \in \mathcal{A}$ is the *activity* associated to event e,
- $\#_{time}(e) \in \mathcal{T}$ is the *timestamp* of event e,
- $\#_{trans}(e) \in TY$ is the *transaction type* of event e, e.g., schedule, start, complete, and suspend.

An event log consists of cases. Typically, execution of a case does not directly influence the execution of other cases, e.g., the way a patient is treated in a hospital does not influence the way other patients are treated. Therefore, we often consider only events for a case in isolation. The events for a case are represented in the form of *complex trace*, i.e., a sequence of unique events.

Definition 2.3.2 (Case, complex trace, complex event log)
A *complex trace* over some event universe $\mathcal{E}$ is a finite sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once, i.e., for $1 \leq i < j \leq |\sigma| : \sigma[i] \neq \sigma[j]$. For all cases $c \in \mathcal{C}, \hat{c} \in \mathcal{E}^*$ is a shorthand for referring to a complex trace of c, such that

- $\{e \in \mathcal{E} \mid \#_{case}(e) = c\} = \{\hat{c}[i] \mid 1 \leq i \leq |\hat{c}[i]|\}$, i.e., all complex events of the same case c are in sequence $\hat{c}$,
- For all $1 \leq i < j \leq |\hat{c}|, \#_{time}(\hat{c}[i]) \leq \#_{time}(\hat{c}[j])$, i.e., all complex events are ordered based on their timestamps.

A *complex event log* $LC \subseteq \mathcal{C}$ is a set of cases. ⌟

For example, the log in Table 2.1 is formalized as $LC = \{1, 2, 3, \ldots\}$. For case $c = 1, \hat{c} = \langle 1023, 1024, \ldots, 1027\rangle$. $\#_{act}(1023) = \mathit{register}$ is the activity associated with event 1023, etc.

Event logs can be used for various types of analysis. Many approaches only require a subset of event attributes. For example, most process discovery algorithms only take activity attribute of events into account, while performance measurement approaches also take time attribute into account. Thus, we define a *classifier* to determine which aspects of importance are taken from event logs.

Definition 2.3.3 (Classifier)
A classifier is a function that maps each event to a representative name used for analysis. For all events $e \in \mathcal{E}$, $\underline{e}$ is the name of the event. ⌟

For example, events $e \in \mathcal{E}$ may be identified by their activity name ($\underline{e} = \#_{act}(e)$) or the pair of activity and lifecycle transition ($\underline{e} = (\#_{act}(e), \#_{trans}(e))$). In this thesis, we use a *default classifier* that maps events to their activity names unless stated otherwise.

In the most of this thesis, we abstract from additional attributes and only use information about recorded activities. Using this abstraction we can formalize traces and event logs as follow:

Definition 2.3.4 (Trace, Event log)
Let $A \subseteq \mathcal{A}$ be a set of activities. A process instance $\sigma \in A^*$, i.e., *trace*, is a sequence of activities. An *event log* $L \in \mathbb{B}(A^*)$ is a multiset of traces. ⌟

The definition of event logs in Definition 2.3.4 abstracts from many details typically stored in event logs. For example, in our formalization there is no unique identifier for process instance, and there is no notion of unique events as there are no event attributes. One can convert complex logs as defined by Definition 2.3.2 into event logs as defined in Definition 2.3.4 using classifiers. We define such transformations as follow.

Definition 2.3.5 (Transforming a complex event log into an event log)
Let $\mathcal{E}$ be a complex event universe and let $LC \subseteq \mathcal{C}$ be a complex event log over $\mathcal{E}$ as defined in Definition 2.3.2. Assume a classifier that returns activity names is chosen. The classifier can be applied to sequences, i.e., $\underline{\langle e_1, e_2, \ldots, e_n \rangle} = \langle \underline{e_1}, \underline{e_2}, \ldots, \underline{e_n} \rangle$. $\underline{LC} = [\underline{\hat{c}} \mid c \in LC]$ is the event log of LC. ⌟

For example, using the default classifier, the formalization of complex event log in Table 2.1 $\underline{LC} = [\langle \mathit{register}, \mathit{lab\ test}, \mathit{decide}, \mathit{payment}, \mathit{archive} \rangle, \langle \mathit{register}, \mathit{lab\ test}, \mathit{decide}, \mathit{surgery}, \mathit{bedrest}, \mathit{decide}, \mathit{payment}, \mathit{archive} \rangle, \langle \mathit{register}, \ldots \rangle, \ldots]$.

Other classifiers can also be used. If we use classifier $\underline{e} = (\#_{act}(e), \#_{trans}(e))$ that maps events to combination of activity name and life-cycle, the obtained log is as follows $\underline{LC} = [\langle (\mathit{register}, \mathit{complete}), (\mathit{lab\ test}, \mathit{complete}), (\mathit{decide}, \mathit{complete}), (\mathit{payment}, \mathit{complete}), (\mathit{archive}, \mathit{complete}) \rangle, \langle (\mathit{register}, \mathit{complete}), (\mathit{lab\ test}, \mathit{complete}), \ldots \rangle, \ldots]$.

In the remainder, we will use whatever notation is most suitable. Furthermore, unless explicitly indicated otherwise, we assume that events are recorded at the moment an activity instance is *completed*, i.e., for all events $e \in \mathcal{E}, \#_{trans}(e) = \mathit{complete}$.

2.4 Petri Nets

Petri nets [119] were the first process modeling languages able to model concurrency. It is still one of the most frequently used notations and the basis for concurrency theory. Various analysis techniques to investigate behavioral and structural properties have been defined in literature. Petri nets are executable and supported by simple yet intuitive graphical notations. In this section, we explain Petri-net related concepts, extensions, and a sub-class of Petri nets that is often used in practice: workflow nets.

2.4.1 Concepts

A Petri net is a bipartite graph consisting of *places* and *transitions*. The state of a Petri net is indicated by the distribution of *tokens* over places and is referred to as *marking*. Transitions can be labeled, e.g., with activities, resources, etc. Figure 2.4 shows a booking process of an online travel agency in Petri net formalism. A customer starts booking by clicking a booking link shown in a browser (start book). Then, he can choose to book a flight ticket (choose flight), hotel (choose hotel), or both. After choosing a flight and/or hotel, the customer has to confirm his selections (confirm). Then, the travel agency books the flight and/or travel tickets (book) while the customer transfers his payment (transfer). A transfer may fail (fail transfer) or succeed (success transfer). A confirmed booking order can be cancelled (cancel) as long as a transfer payment has not been received. The process ends after all historical information is archived (archive).

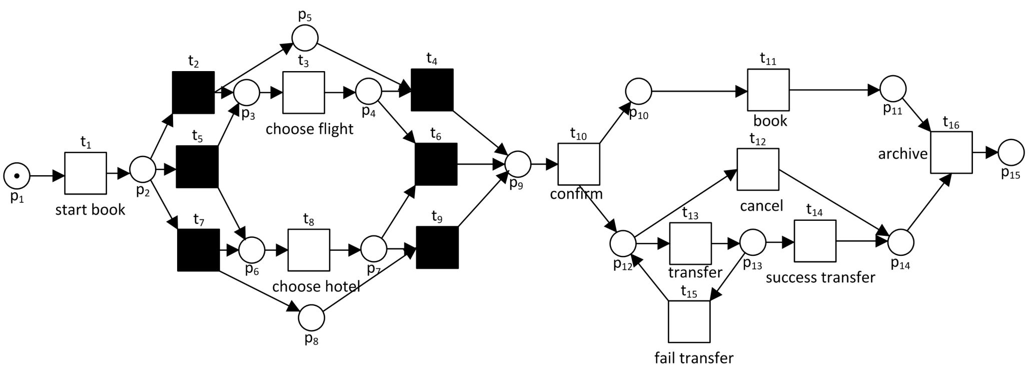


Figure 2.4: A booking process of an online travel agency, shown in terms of a Petri net.

Unless indicated otherwise, non invisible transitions are labeled with activities. We reserve $\tau \notin \mathcal{A}$ as the label of all invisible transitions. For convenience, for any set $A \subseteq \mathcal{A}$, $A^\tau = A \cup \{\tau\}$ denotes the union of set A and $\{\tau\}$. Petri net is formalized as follows:

Definition 2.4.1 (Petri net)
Let $A \subseteq \mathcal{A}$ be a set of activities. A *Petri net* over A is a tuple $N = (P, T, F, \alpha, m_i, m_f)$ where P is the finite set of places, T is the finite set of transitions, $F : (P \times T) \cup (T \times P) \to I\!N$ is a flow relation that returns the weight of arcs, and $\alpha : T \to A^\tau$ is a function mapping transitions to labels.

A *marking*, i.e., a state of the Petri net, is a multi-set of places. $m_i, m_f \in \mathbb{B}(P)$ are the initial and the final marking of N respectively. A transition $t \in T$ is *enabled* at marking $m \in \mathbb{B}(P)$, denoted $(N, m)[t\rangle$ if and only if $\forall_{p \in P}\ F(p, t) \leq m(p)$ hold. $m \xrightarrow{t}_N m'$ denotes the *firing* of an enabled transition t in net N from m that leads to new marking $m' \in \mathbb{B}(P)$, such that $\forall_{p \in P}\ m'(p) = m(p) - F(p, t) + F(t, p)$. A sequence $\varrho \in T^*$ of transitions is a *firing sequence* from marking m to m' if $m \xrightarrow{\varrho[1]}_N m_1 \xrightarrow{\varrho[2]}_N m_2 \ldots \xrightarrow{\varrho[|\varrho|]}_N m'$, abbreviated with $m \xrightarrow{\varrho}_N m'$. We overload notation $(N, m)[\varrho\rangle$ to denote that ϱ is a firing sequence from marking m. Sequence $\varrho \in T^*$ is a *complete firing sequence* if $m_i \xrightarrow{\varrho}_N m_f$. The annotation N of firing sequence ($\to_N$) is omitted if the context is clear. ⌟

The net shown in Figure 2.4 can be formalized as $P = \{p_1, \ldots, p_{15}\}$, $T = \{t_1, \ldots, t_{16}\}$, $A = \{\textit{start booking}, \textit{choose flight}, \textit{choose hotel}, \ldots, \textit{archive}\}$, $m_i = [p_1]$, $m_f = [p_{15}]$, $F(p_1, t_1) = 1$, $F(p_1, t_2) = 0$, …, $F(t_{16}, p_{15}) = 1$. All invisible transitions, such as t_2 and t_4, have the same label, e.g., $\alpha(t_2) = \alpha(t_4) = \tau$.

In principle, multiple transitions may have the same label. We call such transitions *duplicate transitions*. *Invisible transitions* are labeled "τ". Graphically, invisible transitions are colored black as shown in Figure 2.4.

Behavior of a Petri net can be analyzed through the marking reachable from its initial state. We formalize reachable markings in Petri nets as follows.

Definition 2.4.2 (Marking Reachability)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. A marking $m' \in \mathbb{B}(P)$ is *reachable* in N from marking m if there exists a sequence $\varrho \in T^*$ such that $m \xrightarrow{\varrho} m'$. $RS(N, m)$ denotes the set of all reachable markings of N from marking m, i.e., $RS(N, m) = \{m' \in \mathbb{B}(P) \mid \exists_{\varrho \in T^*}\ m \xrightarrow{\varrho} m'\}$ ⌟

The matrix representation of a Petri net, often called the *incidence matrix* of the net, keeps track of the changes on marking while firing transitions. It provides the number of

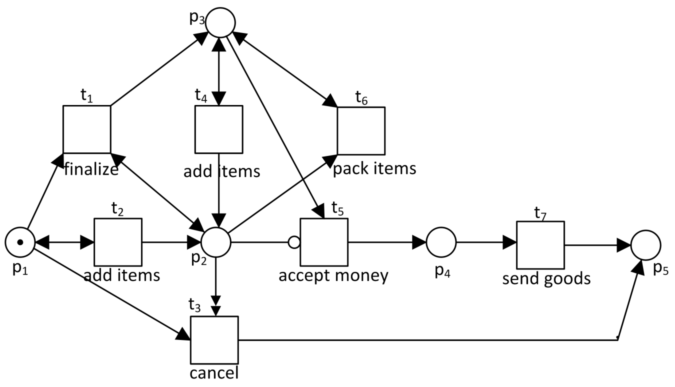


Figure 2.5: An example of a reset/inhibitor net expressing behavior that cannot be modeled in petri net without reset/inhibitor arcs.

tokens consumed and produced by firing transitions. Some analysis techniques for Petri nets, such as transition/place invariants, exploit the incidence matrix for analysis.

Definition 2.4.3 (Incidence Matrix)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. The *incidence matrix* $[\![\mathbf{N}]\!]$ of N is a $|P| \times |T|$ matrix such that forall $1 \leq j \leq |P|, 1 \leq k \leq |T|, [\![\mathbf{N}]\!]_{j,k} = F(T[k], P[j]) - F(P[j], T[k])$. ⌟

A Petri net may still allow for undesirable behaviors such as deadlocks and livelocks. In the remainder of this thesis, we only consider Petri nets whose final marking is reachable from its initial marking. We name such class of Petri nets *easy sound Petri nets*.

Definition 2.4.4 (Easy sound Petri net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. N is an *easy sound Petri net* if and only if $m_f \in RS(N, m_i)$. ⌟

2.4.2 Extension with Reset/Inhibitor Arcs

Petri nets are unable to express complex behaviors such as cancellation and priority. To increase the expressive power of Petri nets, *reset* and *inhibitor* arcs can be added. Petri nets with reset/inhibitor arcs are called *reset/inhibitor nets*. A *reset arc* connecting a place to a transition removes all tokens from the place when the transition fires regardless of the number of tokens originally exist in the place. An *inhibitor arc* connecting a place to a transition does not allow the transition to fire if there is still a token in the place. In the remainder of this thesis, reset arcs are represented graphically by arcs with double arrows, while inhibitor arcs are decorated with a small circle.

Figure 2.5 shows a reset/inhibitor net whose behavior cannot be expressed by a Petri net without reset/inhibitor arcs. The process shows an online transaction process for an electronic bookstore, where transitions are labeled with activities. A customer creates an order by adding as many items as he wants to his cart (add items). After an order is finalized (finalize), all items in the order are packed individually (pack items). All items of an order are sent to customers (send goods) after they are packed and payment for the order is accepted (accept money). The customer may add more items to his cart after the order is finalized (add items), but he can only cancel (cancel) the order as long as it has not been finalized. The process ends when all goods are sent to the customer or the process is cancelled.

Reset/inhibitor nets are formalized as follows:

Definition 2.4.5 (Reset/inhibitor net)
Let $A \subseteq \mathcal{A}$ be a set of activities. A *reset/inhibitor net* over A is a tuple $N = (P, T, F, \alpha, m_i, m_f, r, i)$ where

- $(P, T, F, \alpha, m_i, m_f)$ is a Petri net over A,
- $r : T \to \mathcal{P}(P)$ is a function mapping a transition to its set of reset places, and
- $i : T \to \mathcal{P}(P)$ is a function mapping a transition to its set of inhibitor places

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{B}(P)$ if and only if both $\forall_{p \in P \setminus i(t)}$ $F(p,t) \leq m(p)$ and $\forall_{p \in i(t)}$ $m(p) = 0$ hold. $m \xrightarrow{t} m'$ denotes the firing of an enabled transition t from m that leads to new marking $m' \in \mathbb{B}(P)$, such that $\forall_{p \in P \setminus r(t)}$ $m'(p) = m(p) - F(p,t) + F(t,p)$ and $\forall_{p \in r(t)}$ $m'(p) = F(t,p)$. ⌟

We use the same notation as the one that is already defined in Definition 2.4.1 to denote firing sequences and pre/post of transitions/places. Note that if for all $t \in T, r(t) = \emptyset$ and $i(t) = \emptyset$, N has exactly the same behavior as the Petri net (P, T, F, l, m_i, m_f). Moreover, we extend the notion of easy sound Petri nets in Definition 2.4.4 to Petri nets with reset/inhibitor nets.

2.4.3 Workflow Net

Processes, in particular business or workflow processes, often have a well-defined start and end state. Thus, we consider a subclass of Petri net known as *workflow nets* [168, 169].

Definition 2.4.6 (Workflow net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f, r, i)$ be a reset/inhibitor net over A. N is a *workflow net* if and only if

- There is a single source place $p_i \in P$, i.e., $\{p \in P \mid {}^\bullet p = \emptyset\} = \{p_i\}$,
- There is a single sink place $p_s \in P$, i.e., $\{p \in P \mid p^\bullet = \emptyset\} = \{p_s\}$,
- The source and sink place are the initial and the final marking respectively, i.e., $m_i = [p_i]$ and $m_f = [p_s]$,
- Every node is on a path from p_i to p_s, and
- There is no reset arc connected to the sink place, i.e., $\forall_{t \in T}$ $p_s \notin r(t)$.

⌟

Both nets shown in Figure 2.4 and Figure 2.5 are workflow nets. The reset/inhibitor net shown in Figure 2.5 has a single source place p_1 and a single sink place p_5. It is also easy to see that all nodes are on a path from p_1 to p_5. Furthermore, there is no reset arc connected to p_5. Similarly, the net shown in Figure 2.4 also satisfies all requirements stated in Definition 2.4.6.

Various correctness criteria for workflow nets have been proposed in literature [187]. The most widely used correctness criterion for workflow models is *soundness*, which is defined as follows:

Definition 2.4.7 (Classical Soundness)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f, r, i)$ be a workflow net over A. N is *sound* if and only if

- Option to complete: $\forall_{m \in RS(N,m_i)}\ m_f \in RS(N,m)$,
- Proper completion: $\forall_{m \in RS(N,m_i)}\ m_f \leq m \implies m = m_f$, and
- No dead transitions: $\forall_{t \in T} \exists_{m \in RS(N,m_i)} (N,m)[t\rangle$.

┘

The Petri nets in Figure 2.4 and Figure 2.5 are sound workflow nets, because they satisfy all three criteria mentioned in Definition 2.4.7. The soundness criterion is sometimes considered to be too restrictive in practice. Therefore, weaker notions of soundness have been defined such as weak soundness [104, 105], relaxed soundness [51], and lazy soundness [133]. *Easy soundness* [189] is one of the weakest correctness notions for workflow nets. A workflow net is easy sound if there is at least a firing sequence that leads to proper termination, i.e., the final marking is reachable from the initial marking [187, 189]. Note that this notion of correctness is a specialization of the easy soundness notion formalized in Definition 2.4.4 for workflow nets.

Both nets shown in Figure 2.4 and Figure 2.5 are also easy sound. In both nets, there exists a firing sequence from the initial marking to the final marking.

A workflow net that satisfies a strong correctness notion, e.g., the classical soundness, also satisfies other weaker notions of soundness. For example, all classical sound workflow nets are easy sound, but an easy sound workflow net is not necessarily (classical) sound. The hierarchy of correctness criteria is described in [187] in detail.

2.5 Process Modeling Formalisms

Other than Petri nets, there are alternative languages some of which are often used in practice. We focus on *executable process models*, i.e., process models with semantics that are used to explicitly define the set of allowed traces. In this section, we describe some other existing process modeling languages often used in practice for such purpose. Moreover, we indicate how they relate to Petri nets.

There are two important aspects of process modeling languages that need to be considered when choosing a language to model processes: *expressibility* and *suitability* [91]. *Expressibility* is an objective evaluation of a modeling language based on what modeling problems can be solved by the language considered and what can not. *Suitability* is a subjective evaluation on modeling languages based on how direct are the solutions offered by the languages to modeling problems. In the remainder sections, we provide a short overview on the expressibility and the suitability of existing languages.

2.5.1 BPMN

BPMN [122] is arguably the most used process modeling language in practice nowadays. BPMN was created to provide end-users with the capability of representing their internal business procedures in a graphical manner and communicate them in a standard manner. Figure 2.6 shows a BPMN model of the travel booking process earlier shown as a Petri net (Figure 2.4). The core of BPMN models are tasks, arcs, gateways, and events. *Tasks* in BPMN models are synonymous with transitions in Petri nets. Tasks are labeled and may be connected through directed arcs. *Gateways* are model elements that determine possible paths during execution of a process. Gateways may have *guards*, i.e., a collection of conditions that needs to be satisfied to allow execution of one path. *Events* indicate

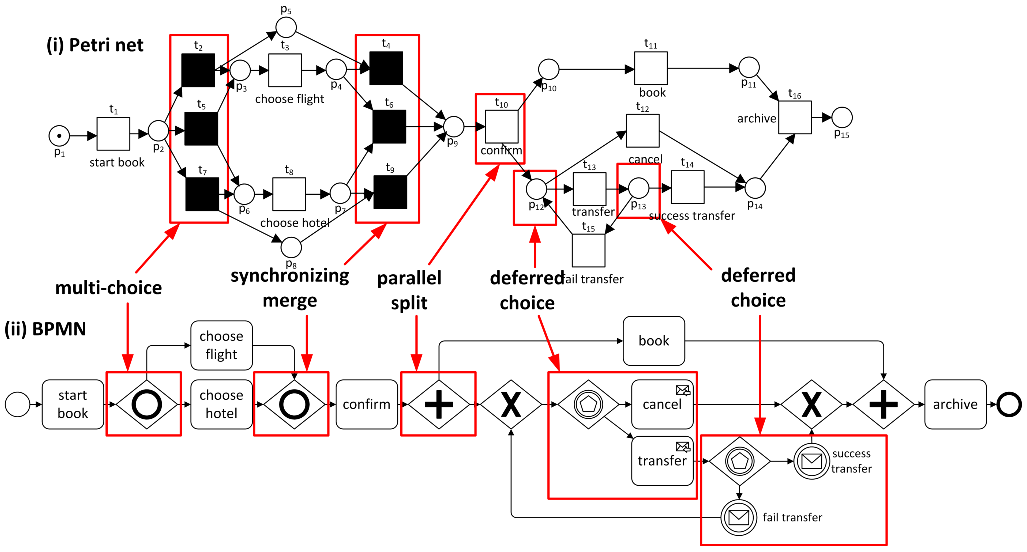


Figure 2.6: A Petri net and a BPMN model that exhibit similar set of traces, annotated with some control-flow patterns that exist in both models.

occurrences of anything in the external environment that may change the state of the process.

Unlike Petri nets, BPMN provides abundance of notations to represent complex patterns. For example in Figure 2.6, both multi-choice and synchronizing merge patterns are each represented by a dedicated gateway. There are many ways of expressing the same behavior in BPMN. Figure 2.6 shows two alternatives to represent the same deferred choice pattern: using tasks with type “receive”, or using events.

Despite of its popular use, BPMN also has some drawbacks. States are not explicitly defined in BPMN. Therefore, some state-based patterns such as the milestone pattern cannot be expressed in BPMN [209]. Furthermore, despite of the abundance of notations, only small subset of them are really used in practice [211]. Some notations are ambiguous as shown in [57], thus models with such notations cannot be analyzed without making further assumptions. The work in [57] even suggests that analysis on BPMN models should be performed after translating them to Petri nets.

2.5.2 YAWL

YAWL is a process modeling language based on Petri net that was developed to be both *expressive* and *suitable* [184]. YAWL is based on Petri nets, therefore it inherits the advantages of Petri net-based language such as [167]:

- Formal semantics despite the graphical nature,
- State-based instead of (just) event-based, and
- Abundance of analysis techniques.

YAWL extends classical Petri net language by supporting directly non-trivial workflow patterns, such as advanced branching and synchronization (e.g., OR-split, OR-join), multiple instances, and cancellation patterns. YAWL models consist of tasks, conditions, arcs, and possibly cancellation regions. Tasks and conditions in YAWL models are the same as transitions and places in Petri nets. Two special types of conditions, i.e., input and output

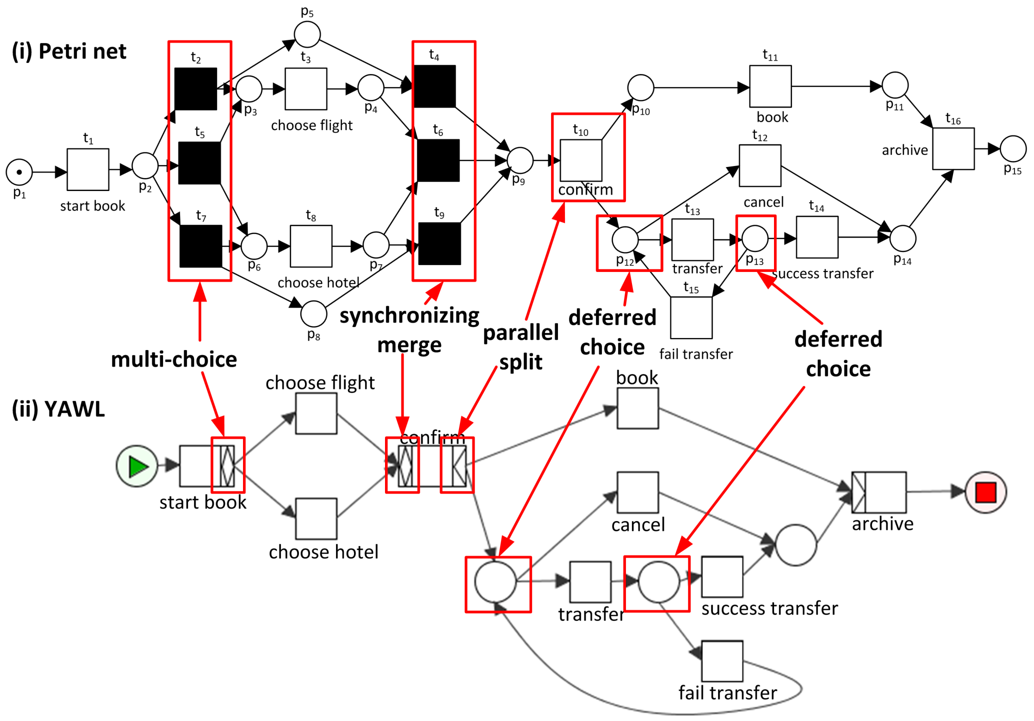


Figure 2.7: A Petri net and a YAWL model that exhibit similar set of traces, annotated with some control-flow patterns that exist in both models.

condition, must exist exactly once in a process model. These indicate the start and end of process respectively. Figure 2.7 shows the net shown in Figure 2.12 and a YAWL model that allows similar behavior. Notice that advanced control-flow patterns are succinctly encoded as part of tasks, and conditions can be omitted for clarity. We refer to [83, 184] for details about YAWL.

2.5.3 Causal Nets (C-Nets)

Process modeling languages typically have elements that represent activities, but often they also have extra elements to specify the relation between activities (semantics). For example, a Petri net requires places to represent states of the process and to model the control-flow. YAWL also has conditions to represent the state of process. BPMN has gateways and events. All of these extra elements do not leave a trace in event logs, i.e., no events are generated by occurrence of elements other than the ones representing activities. Thus, given an event log, process discovery techniques must also “guess” the existence of such elements in order to describe the observed behavior in the log correctly. This causes several problems as the discovered process model is often unable to represent the underlying process well, e.g., the model is overly complex because all kinds of model elements need to be introduced without a direct relation to the event log (e.g., places, gateways, and events) [176]. Furthermore, generic process modeling languages often allow for undesired behavior such as deadlocks and live-locks.

Causal nets are a process model representation tailored towards process mining [176].

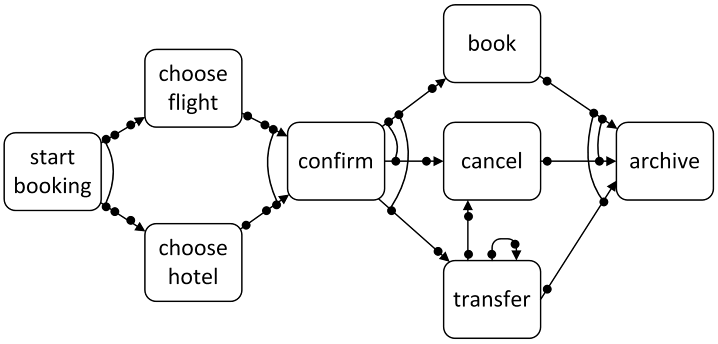


Figure 2.8: A causal net with similar behavior as the Petri net in Figure 2.4.

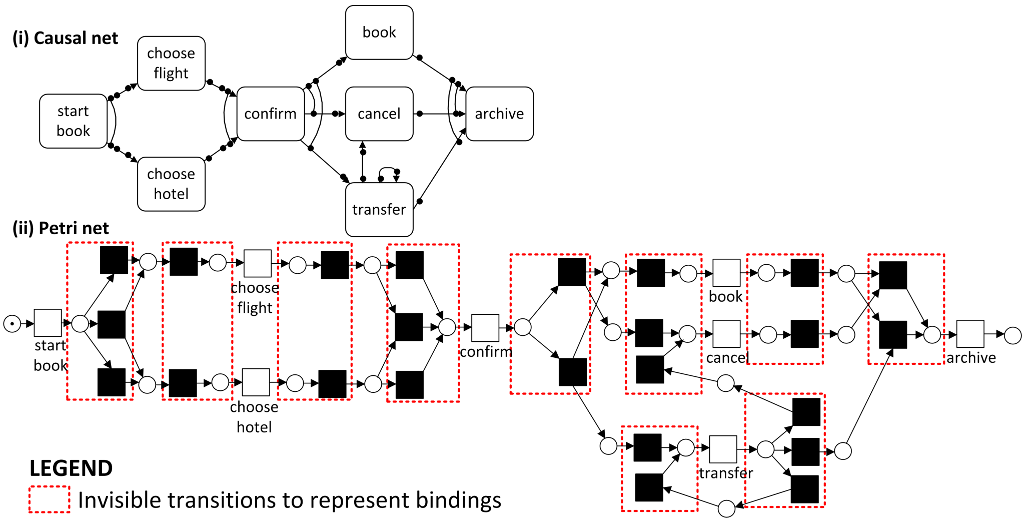


Figure 2.9: (i) The causal net in Figure 2.8 and (ii) a Petri net constructed from (i) that allows all traces of (i) and some additional traces.

Nodes in a causal net represents activities and arcs represent dependencies between them. A causal net has a start and an end activity. Semantics of activities are shown by their input and output bindings.

Consider, for example, the causal net depicted in Figure 2.8. The causal net shows the same booking process in an online travel agency as the one shown by the transition system in Figure 2.12. There are three output bindings for activity start booking: the binding with activity choose flight, the binding with choose hotel, or the binding with both. When an activity is executed with a binding, an obligation is created according to the binding. Suppose that start booking is executed with binding choose flight, then there is a pending obligation to do perform choose flight. Only after choose flight is executed with input binding start booking, the obligation is removed. A causal net describes behavior that starts from the start activity and ends with the end activity without any pending obligations. Note that complex control-flow patterns such as cancellation and multiple instances must be encoded explicitly as bindings of activities. Thus, causal nets are less suitable to express processes with a complex control-flow.

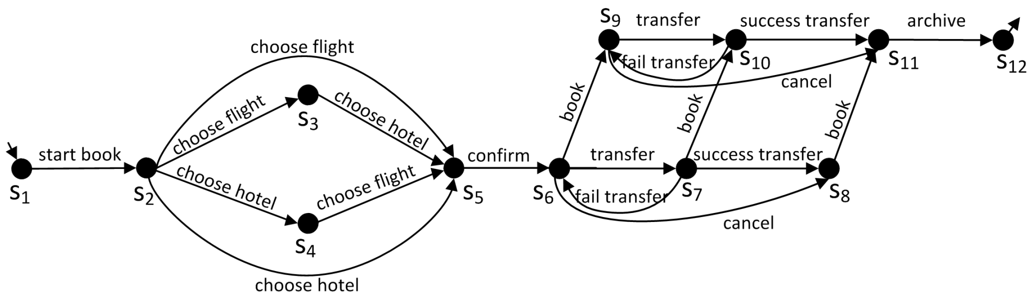


Figure 2.10: A transition system of a booking process in an online travel agency.

For all causal nets, there is a Petri net that allows the same set of traces with possible additional behavior. We refer interested readers to [171] for details on converting a causal net to a Petri net. Figure 2.9 shows a Petri net constructed from the causal net in Figure 2.8 using the conversion rules in [171]. Note that many invisible transitions are needed to represent bindings in the original Petri net. This shows the advantage of using causal nets to model activities with complex control-flow between activities. Similar to BPMN models, causal nets have no explicit notion of state.

2.5.4 Labeled Transition Systems

Transition systems are one of the most basic process modeling notations. Figure 2.10 shows a transition system of a booking process in an online travel agency. As shown in Figure 2.10, a transition system consists of *states* and *transitions* that connect the states. Note that the notion of transitions in transition systems is different than the notion of transitions in Petri nets. States are represented by black circles and the arcs connecting the circles represent transitions. A transition of a transition system is labeled, typically with an activity. Furthermore, a transition system has an *initial state* and a *final state*. Graphically, the initial state of a transition system has a small incoming arc and its final state has a small outgoing arc. In Figure 2.10, state s_1 and s_{12} are the initial and the final state respectively.

In this thesis, we consider *labeled transition systems*. Labeled transition systems can be formalized as follows:

Definition 2.5.1 (Labeled Transition System)
A *labeled transition system* is a tuple $LTS = (S, TR, LB, s_i, s_f)$ where S is the (possibly infinite) set of *states*, LB is the set of *labels*, and $TR \subseteq S \times LB \times S$ is the set of *transitions* between states. $s_i, s_f \in S$ is the *initial* and *final* state respectively. ⌟

Transition systems are very expressive. Many process models with executable semantics can be mapped onto a transition system [171]. Thus, analysis techniques and notions defined for transition systems can be easily related to other languages such as BPMN, BPEL, EPC, and Petri net. A transition system can be translated to a Petri net that allows for the same set of traces. For example, Figure 2.11 shows a Petri net that allows for the same set of traces as the labeled transition system in Figure 2.10.

There is, however, a drawback to use transition systems for modeling processes. Concurrency cannot be expressed succinctly in transition systems. In all transition systems, all possible interleaving between parallel transitions activities must be encoded explicitly.

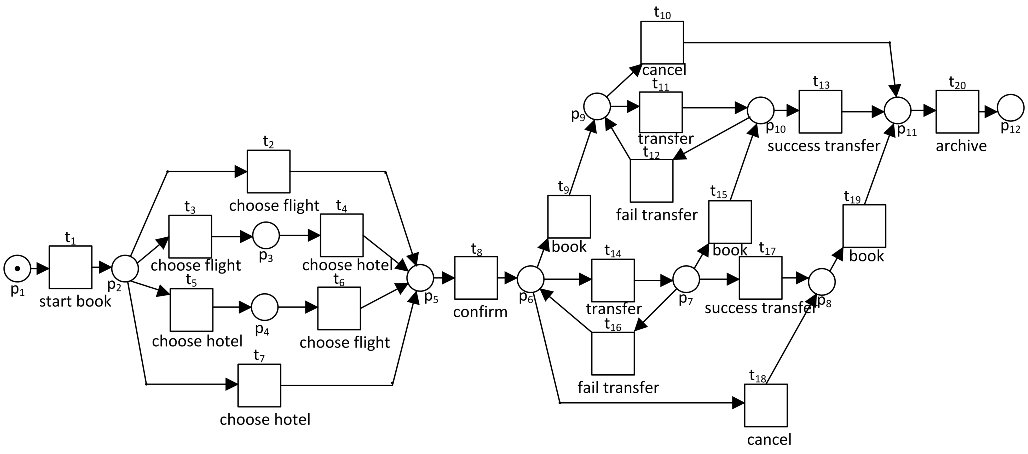


Figure 2.11: A Petri net that allows for the same set of traces as the labeled transition system in Figure 2.10.

Therefore, despite being expressive, transition systems are often *not suitable* to model business processes.

A reset/inhibitor net whose initial marking is reachable from its initial marking can be translated into a transition system that allows for the same set of traces. Such a translation can be formalized as follows.

Definition 2.5.2 (Labeled transition system of a reset/inhibitor net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f, r, i)$ be a reset/inhibitor net over A where $m_f \in RS(N, m_i)$. The labeled transition system of N, denoted $TS(N) = (S, TR, LB, s_i, s_f)$ is a tuple where

- $S = RS(N, m_i)$, i.e., the set of all reachable states from the initial marking m_i,
- $LB = T$, i.e., the set of labels is the set of transitions of N,
- $TR = \{(m, t, m') \in S \times T \times S \mid m, m' \in S \wedge t \in T \wedge m \xrightarrow{t} m'\}$ is the set of all state transitions,
- $s_i = m_i$ is the initial state, and
- $s_f = m_f$ is the final state.

┘

Figure 2.12 shows the transition system of the net in Figure 2.4. The set of labels in the transition system is the set of Petri net transitions. The transition system is isomorphic to the transition system in Figure 2.10. Furthermore, if we replace all Petri net transitions in Figure 2.12 with their activity label, we obtain the same transition system as the one shown in Figure 2.10. Note that the transition system of a Petri net is *deterministic*, because firing a transition from a marking in the net always lead to a unique marking. Furthermore, for all transition systems (S, LB, TR, s_i, s_f), (S, LB, TR) can also be viewed as a directed graph.

In literature, there are various approaches to convert a labeled transition system to a Petri net. Given a transition system, the work in [40, 56, 61] shows that the so-called theory of regions can be used to construct a Petri net whose labeled transition system is bisimilar to the given one. "Regions" are defined as sets of states in the transition system and then translated to places in the net. The construction of a Petri net from a given

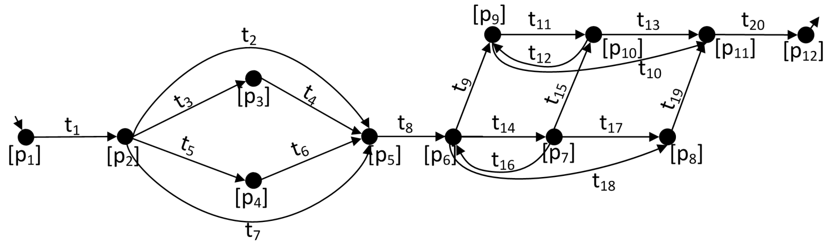


Figure 2.12: The labeled transition system of the net shown in Figure 2.11. The transition system is isomorphic to the transition system in Figure 2.10.

transition system is often called *synthesis*. This shows that a Petri net can be converted to a labeled transition system and vice versa.

2.5.5 The Process Modeling Formalism Used in This Thesis

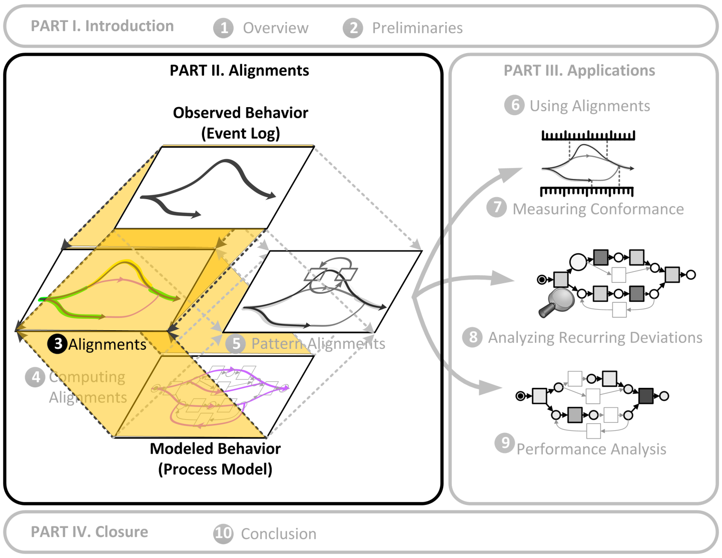
In this thesis, we take a pragmatic approach to select the most suitable process modeling formalism based on our objective. We choose Petri nets and their extensions (reset/inhibitor nets) as our process modeling formalism whenever we need a formal and/or expressive model. Petri nets have unambiguous semantics, a standard graphical notation, and they are supported by abundance of analysis techniques. Almost all executable process models can be converted to Petri nets with similar behavior (especially when allowing for reset and inhibitor arcs). Furthermore, many other process modeling languages can be translated to transition systems and transition systems can be easily converted into Petri nets (and vice versa). Hence, Petri nets are good representatives for existing process modeling languages.

Part II

Alignments

Chapter 3

Relating Event Logs to Models



3.1 Introduction

Business processes have been studied since the end of 18th century when Adam Smith (1723-1790) showed the advantages of the division of labor. However, only around the 1950-ties Information Technology (IT) started to influence business processes and their management. IT may provide support for business processes beyond traditional approaches that rely on human resources. Over time, business processes have become more complex and may even span over multiple organizations. Process models assist organizations to manage such complexity by offering insights into process executions. Furthermore, common agreement of required interactions between organizations is crucial in conducting cross-organizational processes. Process models offer easily understandable yet reasonably precise documentation of business process and thus help to reduce the risk of misinterpretation. As a consequence, many organizations nowadays have process models of their business processes.

Process models may serve many different purposes. In BPM, there are three main purposes of process models [83]. First of all, models may aim at providing *insights*. Models that serve this purpose are made to represent reality, i.e., they are *descriptive* in nature. Second, process models may be used to *analyze* the system and/or its processes. The type of analysis that can be performed depends on the models used. Typically, formal, unambiguous models offer more analysis capability than non-informal ones [167]. Third, models are used to *enact* processes. These models are *prescriptive* in nature, i.e., they describe the way processes should be executed. For example, reference models in information systems such as SAP R/3 [89] and ARIS [153] act as guidelines on how processes should be performed. However, they do not restrict executions and people may deviate from these guidelines.

Regardless of being *descriptive* or *prescriptive*, process models have to be *aligned* with reality. If a process model allows for behavior that never occurred in reality, insights and analysis results obtained from the model may be misleading. As mentioned in Section 1.2, the abundance of process-related data nowadays allows us to align observed behavior in reality with modeled behavior in process models. In systems where process executions are strictly enforced by process models, there is no need to create such alignments. However, some degree of flexibility to deviate from prescribed process models is often desired. Furthermore, models may exist independently from event logs, i.e., models may exist only on paper and events may have no reference to process models.

In literature, there exists many process modeling formalisms as shown in Chapter 2. We take a pragmatic approach to select the most suitable process modeling formalism based on our objective. We choose Petri net and its extensions (reset/inhibitor nets) as our process modeling formalism whenever we need a formal and/or expressive model. Petri nets have unambiguous semantics, a standard graphical notation, and they are supported by abundance of analysis techniques. Moreover, almost all executable process models can be converted to Petri nets with similar behavior (especially when allowing for reset and inhibitor arcs).

Given an event log and a Petri net, a fundamental challenge is to correlate occurrences of events in the log to transitions in the net, i.e., *replay* the events on the net. Replay approaches must be robust to handle peculiarities of event logs and Petri nets. They have to be *scalable*, i.e., they are able to handle sufficiently large and complex event logs and Petri nets. Furthermore, replay results must also provide additional insights that can be analyzed further.

In this section, we list some important challenges that must be tackled by replay approaches. To illustrate the challenges, we use the Petri net shown in Figure 3.1. The net is an easy sound workflow net, but it is not a sound net (see Definition 2.4.7 and Definition 2.4.4). The net shows a claim handling process in an insurance company. The process starts when a claimant files a claim (claim). A decision is made by a manager whether the claim will be audited (audit) or not. If the manager decides to audit the claim, some additional information may be requested from the claimant (request data). A decision whether the claim is accepted (accept) or rejected (reject) is based on this data. If the claim is accepted, the company pays some money to the claimant via a dedicated account (pay). Then, a notification may be sent to the claimant about the acceptance through post (send post), mail (send mail), or not at all, depends on the policy that applies to the claimant. The process ends when an administration officer archives the whole claim handling (archive).

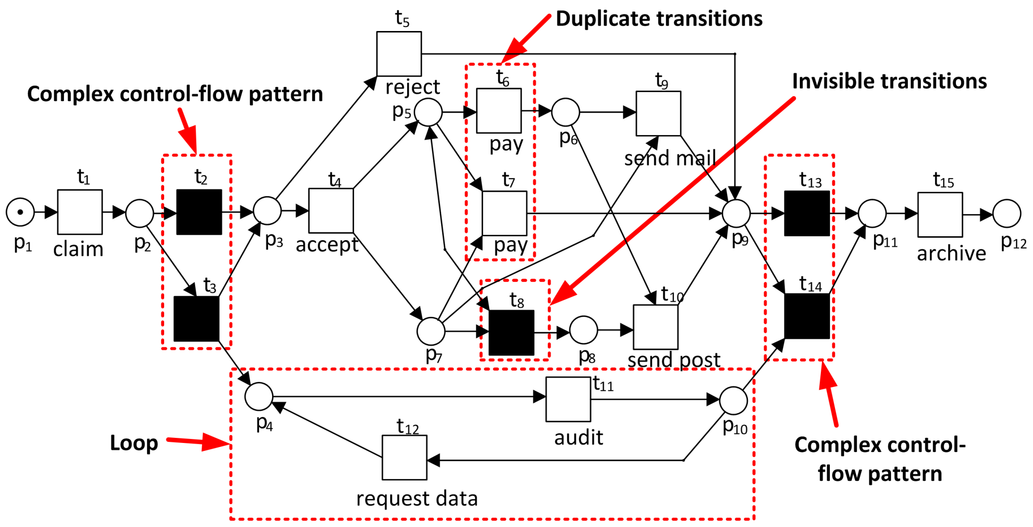


Figure 3.1: A potentially challenging Petri net for traditional replay approaches.

Using the net as a running example, we illustrate the following challenges of replay approaches mentioned in literature [9, 48, 145, 175]:

1. **Duplicate transitions.** A Petri net may have multiple transitions with the same label. In Figure 3.1, both t_6 and t_7 have label pay. Suppose that both t_6 and t_7 are both enabled, e.g., the marking of the net is $[p_5, p_7]$, and an event with activity attribute pay occurred. A good replay approach should be able to tell to which transition the event belongs to.
2. **Invisible transitions.** In the presence of transitions without label, i.e., invisible transitions, a replay approach should not mistakenly consider a perfectly fitting execution as deviating and vice versa. For example, suppose that a firing sequence $\langle t_1, t_3, t_4, t_8, t_6, t_{11}, t_{10}, t_{14}, t_{15}\rangle$ is performed for a case. Since some transitions in the sequence are invisible, the trace of the case is $\langle claim, accept, pay, audit, send\ post, archive\rangle$. A replay approach should be able to consider that this trace is perfectly fitting the net, i.e., it can be fully reproduced by the net without any remaining tokens. In this example, the approach should be able to identify the occurrence of invisible transition t_8 such that occurrence of *send post* in the trace is not accounted as deviation. As another example, take a non-fitting trace $\langle claim, accept, send\ post, pay, archive\rangle$. A replay approach should be able to identify that this trace is a non-fitting trace as according to the net, activity send post must not occur before activity pay.
3. **Complex patterns.** A replay approach should be able to deal with complex control-flow patterns that may exist in process models. For example, invisible transitions t_2 and t_3 in Figure 3.1 express a choice between only doing acceptance/rejection for a claim or also execute auditing step. Suppose that a trace is a sequence $\langle claim, reject, archive\rangle$, i.e., the trace only follows the top branch control-flow of the model. The replay approach should not mistakenly consider the sequence as deviating because there is no activity audit in the trace.
4. **Loops.** Behavior that a Petri net exhibit can be infinite if the net has loops. For example, the net in Figure 3.1 allows for infinitely many iterations of audit followed

by request data. Given a trace of a case and a Petri net that allows for loops, a replay approach should still be able to map occurrences of events in the trace to transitions in the net.

5. **Sensitivity to deviations**. In cases where observed behavior is not allowed according to a Petri net or vice versa, a replay approach must not be sensitive to all types of deviations, e.g., deviations that occur early should not influence the mapping of events that occur later in the process execution. For example, suppose that a trace of a case of the net in Figure 3.1 is $\langle claim, claim, audit, accept, pay, send\ post, archive\rangle$. It is easy to see that two consecutive occurrences of activity claim are not allowed according to the net. A replay approach should not stop after the first deviation, i.e., the alignment should map other events after the deviations, such as audit, accept, pay, send post, and archive.
6. **Support models with strict semantics**. Replay result is used as basis for various analysis based on observed and modeled behavior. Therefore, it is important that the replay result is concise and unambiguous. This is clearly possible in case of process modeling languages with strict semantics such as Petri nets. Process modeling languages with less strict semantics, e.g., Fuzzy models [75], SPDs [193], and process maps [99], allow for behavior not represented by their nodes and arcs, thus they complicate further analysis.
7. **Diagnostics**. A replay approach should be able to provide information beyond whether traces can be reproduced by process models. For example, if the behavior observed in an event log is not allowed according to a Petri net, the replay result should provide diagnostic information explaining why the process deviates.
8. **Scalable**. With the availability of big data, it is increasingly important that a replay approach should be able to deal with large logs and models. Thus, computation complexity and memory requirements also need to be taken into account when evaluating replay approaches.

3.2 Related Work

In this section, we evaluate several replay approaches using the list of replay challenges given in Section 3.1. We provide an overview of each approach and list its strengths and weaknesses.

3.2.1 Token-based Replay

The token-based replay approach proposed by Rozinat et al. [151] measures the conformance between an event log and a Petri net. Similar approaches are also used for the evaluation of process discovery algorithms [47, 204]. Given an event log and a Petri net, token based-replay takes each trace in the log in isolation and fire transitions sequentially according to the ordering of events in the trace. If a transition should be fired according to an event in a trace but it is not enabled, enabled invisible transitions are fired to enable the transition. If there are no such invisible transitions, missing tokens are added to enable the transition. All added tokens are recorded. Together with the number of remaining tokens left after all traces are replayed, the amount of added tokens is used to measure conformance between the log and the net.

If an event can be associated to more than one enabled transition, i.e., there are multiple transitions enabled with the same label as the activity attribute of the event,

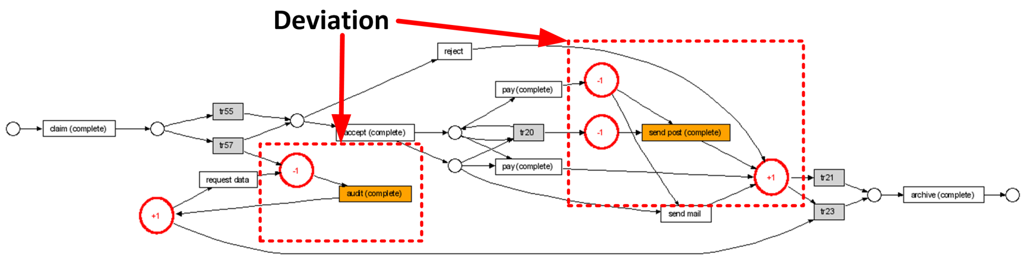


Figure 3.2: Screenshot of replaying trace $\langle claim, accept, pay, audit, send\ post, archive\rangle$ on the net in Figure 3.1 using the token-based replay in ProM 5.2.

state space analysis using heuristics is performed to find out which of the candidate transitions enables the transition of the direct successor of the event in the trace. This process is repeated until there is one candidate left. If there are multiple candidates and no more events are left in the trace, transitions are chosen randomly. The same approach is also applied if a transition that should be fired according to an event is not enabled and multiple invisible transitions are enabled instead.

The approach was implemented and publicly available in the ProM 5.2 framework[1]. Due to the heuristic nature of the approach, its computation complexity is relatively low but sometimes still time consuming in cases where state space analysis needs to be performed. Furthermore, it also provides diagnostics. The number of missing tokens and remaining tokens can be projected onto the original Petri nets to provide diagnostics on where deviations are and what are the cause of deviations. Furthermore, the implemented approach in ProM 5.2 also provides options to show deviations per case. The approach has shown to be useful in many case studies, e.g., [46, 151, 170, 179, 180].

However, heuristics in the approach may lead to misleading results when dealing with Petri nets with invisible and duplicate transitions. Figure 3.2 shows the result of replaying trace $\langle claim, accept, pay, audit, send\ post, archive\rangle$ on the net in Figure 3.1 using the implemented token-based replay in ProM 5.2. Sequence $\langle t_1, t_3, t_4, t_8, t_6, t_{11}, t_{10}, t_{14}, t_{15}\rangle$ is one possible firing sequence of the net that generates the trace. Thus, there is no deviation on the trace. However, as shown in the figure, the occurrence of $audit$ and $send\ post$ are considered as deviations. The approach fails to identify the occurrence of invisible transitions t_3, t_8, and t_{14} correctly. In [196], vanden Broucke et al. list several strategies to replay the positive events of a trace on a Petri net, similar to [151]. However, none of them guarantees that the “optimal” results, i.e., the replay that minimize the number of deviations without introducing new behavior, are always obtained.

The addition of missing tokens may allow extra behavior that cannot be performed in the original Petri net. Thus, the measurement of deviations can be misleading. Moreover, in cases where deviations occur frequently, places may be flooded with tokens. In such cases, diagnosis results are unreliable and it is impossible to find the root cause of deviations. Moreover, the fitness estimates are too positive since many transitions remain enabled because of unused tokens.

[1]see `http://processmining.org`

3.2.2 Replay with Artificial Negative Events

Given an event log, the approach proposed in [49, 68] appends the log with artificial negative events to discover a Petri net from positive and negative events. To evaluate the quality of discovered net, the approach also proposes a way to replay events in the log similar to the token-based replay of Section 3.2.1. Events of each trace in the log are parsed independently from events of other traces. When a transition that should be fired according to an event in a trace is not enabled, it is *forced* to fire. In case of multiple enabled duplicate transitions, the transition that enables the next positive event in the trace is fired. If both transitions (do not) enable the next positive event, a random choice is made. Invisible transitions are handled in a similar way. If there are multiple enabled invisible transitions, the one that enables the transition(s) of the next positive event is chosen. If none of them enables the transition(s) or both enable the transition(s), a random choice is made. Negative events are used to quantify the specificity of the discovered net with respect to the log. If the net allows for the execution of a negative event, the specificity of the net is decreased. However, negative events do not influence the way the positive events are mapped to transitions.

The proposed replay approach in [68] uses heuristics to enable currently parsed event in the trace without necessarily considering successors of the event, i.e., the analysis is performed *locally*. As a consequence, long term dependencies between transitions may not be identified correctly from the trace. This may provide misleading result when dealing with both duplicate and invisible transitions, the same as the token-based replay. This implies that the approach also has problems to deal with complex control-flows that require invisible transitions in the Petri net. Different than the token-based replay, the replay proposed in [68] only considers one-level (i.e., a look ahead of one step) to select a duplicate and invisible transition for an occurrence of an event. Therefore, the computation complexity of this approach is linear in the length of traces and hence, less complex than the token-based replay.

3.2.3 Hidden Markov Model Conformance Checking

A *Markov model* is a stochastic model of a process that consists of a set of states and a probability distribution describing the likelihood to move from one state to another. A Markov model has a unique property that for each state, the conditional probability distribution of its future states depends only upon the current state. Each state of a Markov model corresponds to an observable event. Hidden Markov Models (HMMs) [136] are an extension of Markov models where a state may emit more than one type of observed events. Each state has a probability to emit an event of a particular type. Thus, given a HMM and a sequence of emitted events, there can be more than one path in the HMM that produces the same sequence of events. HMMs are used for various type of analysis, such as speech, pattern, and gesture recognition, part-of-speech tagging, and bioinformatics.

Given a sequence of emitted events and a HMM, one of the fundamental problems whose solution exists in literatures is finding a sequence of states in the HMM that has the highest probability to generate the sequence. Rozinat et al. [152] transform the problem of measuring the quality of a discovered process model (in Petri net formalism) from an event log into the problem of finding such sequences in a HMM. First, the Petri net is converted to a HMM, and the log is treated as a set of sequences of emitted events. Using existing technique such as the *Viterbi algorithm* [64], the sequence of HMM states that has the highest probability to generate the sequences can be identified. Using such sequences, occurrences of events in the log can be easily related to transitions in the original Petri net.

The Viterbi algorithm identifies a sequence of states with the highest probability to generate a given sequence of emitted events, regardless whether there are more than one state that emit the same events. Therefore, this approach is robust to duplicate transitions. However, the approach has several limitations. First, the proposed conversion method is limited to a subset of Petri nets that does not allow any parallelism (i.e., *state machines*). Furthermore, all states in the constructed HMM must be connected to address all possible deviations between traces and Petri nets. Since the computation complexity of the approach is exponential in the number of states and the number of transitions between them, the approach may not be able to handle sufficiently large nets and event logs. All states in HMM emit events, so the approach does not support invisible transitions.

3.2.4 Posteriori Data Purpose Control Analysis

The comparison between event logs and process models is also useful to check possible data misuse in security domain. Petkovíc et al. [114] proposed a framework to check compliance of executed process in event logs to data protection policies described in form of BPMN. The approach uses the COWS formalism [94], a foundational language for service-oriented computing that combines process-calculi and WS-BPEL.

Given a BPMN model and a trace (also called *audit trail* in [114]), the approach first converts the model into a set of COWS specifications. A set of COWS specifications can be represented as a labeled transition system where each state contains a collection of tasks that can be executed from the state, i.e., *active tasks*, and transition labels are the values of attributes observable from events in the trace. Events in the trace are paired with transitions in the transition system. If there is more than one transition that can be paired with an event, both states are used as possible current states. If there is no transition that corresponds to an event, a deviation is identified and the rest of the events are ignored.

All transitions in the labeled transition systems must be observable from events. Therefore, to deal with invisible tasks, separate states are created for each possibility of performing invisible tasks (or sequences of invisible tasks) from a current state. For example, if two invisible tasks t_1 and t_2 are enabled, two states are created where the first state contains active tasks if t_1 is performed, and the other contains active tasks if t_2 is performed. Note that a BPMN model may allow infinite sequences of invisible tasks (loop of invisible tasks). Therefore, this approach excludes BPMN models with loops of only invisible tasks. The approach keeps track of all possible current states until there are no more events in the trace that can be mapped or deviations occur. When an event in the trace deviates from the model, the rest of the events (i.e., successors of the event) in the trace are ignored.

By keeping all possible current states, this approach tries all possible mappings before concluding that a deviation has occurred. Thus, it is robust to invisible/duplicate tasks and complex control-flow patterns. However, this also means that it requires extra memory to store all possible current states. Furthermore, the approach ignores all events that occur after the first deviation. Therefore, it is sensitive to deviations and only provides limited information to be used for further diagnosis.

3.2.5 Measuring Privacy Compliance

Another example from the security domain is the approach by Banescu et al. [16]. They propose an approach to measure privacy compliance with respect to process specifications. Specification of a process is given as a process model. Suppose that the model is described in Petri net formalism, each transition in the net is labeled with an activity. For all transitions in the net, data items that should be accessed and roles of resources that may perform the transitions are explicitly specified. Such a specification is compared against observed behavior in form of a sequence of events, i.e., trace, where all events have the following attributes: activity, resource, and accessed data items.

To quantify compliance between a trace and a Petri net, the approach picks a complete running sequence allowed according to the net that is the most similar to the trace. Similarity between a running sequence and a trace is computed in the same way as computing the Levenshtein distance [98] between them. However, instead of using the distance of 1 and 0 as in computation of Levenshtein distance, the approach uses a cost function that maps pairs of event in the trace and transition in the model to non-negative values. For each pair of event and transition, the value returned by the cost function depends on (1) the reputation of the resource that performs the event, (2) the semantic distance between the label of the transition and the activity attribute of the event, (3) the semantic distance between the role of the resource performing the event and the role allowed to perform the transition, and (4) data authorization.

The approach can easily handle Petri nets with either duplicate or invisible transitions. However, given a trace and a Petri net, one of the challenges of this approach is to identify which complete firing sequence of the net is most similar to the observed trace. In [16], all complete firing sequences of the net are computed in advance and the "real distance" between the trace and the net is the minimum distance that one can obtain from all possible complete firing sequences. If the net has infinitely many complete firing sequences, e.g., if loop exists, this approach cannot be used (without leaving out potential solutions).

3.2.6 Understanding the Behavior of Agents in Business Processes

In [63], Ferreira et al. propose a technique to map events at a low level of granularity to a process model whose tasks (i.e., transitions in Petri net terms) are at a higher level of granularity. Figure 3.3 shows an example of a hierarchical process models. The model at the higher level of granularity is called the *macro-level model*, while the models at the lower level of granularity are called *micro models*. Suppose that a macro-level model is given and multiple *agents* perform the tasks in the model. The agents leave a trace of executions in low level of granularity (i.e., a *micro-sequence*). The proposed technique takes the micro-sequence and the model to construct a set of micro-models that best describe the behavior of agents. Note that in the work of [63], all models are given in the form of Markov chains.

Given a macro-level model and a micro-sequence, [63] uses a procedure based on expectation-maximization to construct a set of micro-models. First, a random sequence of the macro-level model is chosen to represent the micro-sequence, and a sequence of tasks in the macro-level model is generated with the same length as the micro-sequence. This sequence is called a *macro-sequence*. Then, the macro-sequence is used to estimate a set of micro-models. The quality of the set is measured based on its probability to generate the original micro-sequence. The set of micro-models is again used to obtain

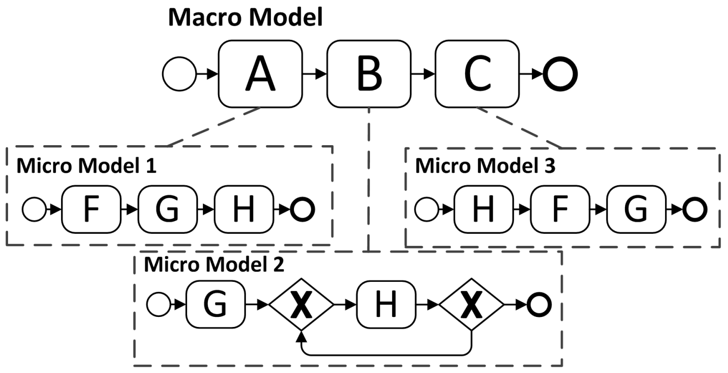


Figure 3.3: A hierarchical process model in BPMN notation.

a better estimate of the macro-sequence. The process continues iteratively until both macro-sequence and set of micro-models converge.

Markov chains are one of process modeling formalisms with a rather strict semantics, thus the result of this approach also provide a useful insights for further analysis. However, this approach assumes that micro-sequences perfectly fit into macro-level models. Hence, the approach is not able to deal with noise and does not provide any diagnostics in case of deviations. The iterative nature of the approach also implies that it is computationally expensive, especially when dealing with models that have complex flow-patterns, loops, and duplicate/invisible tasks. This may limit the applicability of the approach to deal with complex logs and models.

3.2.7 Comparing Event Streams with Model Streams

Cook et al. [39] propose an approach to compare event streams, i.e., traces, with process models to quantitatively measure their similarity. The similarity between a trace and a process model is quantified based on the number of insertions and deletions one needs to apply such that the trace can be transformed into a stream of events allowed by the model, i.e., *model stream*.

Given a trace and a process model, the approach pairs occurrences of events in the trace to tasks in the process model (i.e., transitions in Petri net terms) with the same activity label using state-space search techniques. If such mapping cannot be performed, either an extra event is inserted into the trace or the currently processed event is removed from the trace. A state in a state-space may represent one of the following operations: matching an event with a task, deleting an event in the trace, or inserting extra events. Such a state-space is build incrementally from the beginning of the trace and the initial state of the model until both the end of the trace and the termination of the model is reached. Since the search process only moves forward, the whole state space is described in terms of a tree.

To avoid the well-known state-space explosion problem, the approach uses best-first strategy to explore state-space efficiently. The best-first search uses a priority queue of states to be evaluated, and always evaluates the lowest-cost state on the priority queue. Furthermore, it may also use a heuristic estimation function that estimates the distance from all states to goal states to further "guide" the state-space exploration without sacrificing results, i.e., if the estimation always underestimates the true cost, best-first search is guaranteed to find a solution with minimum cost. The approach is extended in [38] to also consider time aspects.

The results of comparison between two streams proposed in [39] tend to be robust to both duplicate and invisible tasks, loops, and complex control-flow patterns that may exist in process models. Moreover, deviations between traces and process models are explicitly shown by the insertions and deletions that need to be performed. However, this robustness comes with price of computation complexity. To increase its applicability in practice, an underestimation function and several pruning approaches are proposed in [39]. However, none of them guarantee that the same comparison result as the one obtained without such shortcuts, i.e., the one with minimum number of deviation, can be obtained. Nevertheless, this approach is most related to the work performed in this thesis.

3.2.8 Checking Executability of Scenario in Specification

Specifications of distributed systems are often provided in form of scenarios. Juhás et al. [88] propose an algorithm with polynomial complexity to check whether scenarios in form of Labeled Partial Orders (LPOs) can or cannot be executed in a Petri net. An LPO is a directed graph whose nodes are labeled with transitions and arcs between the nodes represent ordering between nodes. An LPO is not necessarily a connected graph. Therefore, LPOs are more expressive than merely sequences of transitions (i.e., occurrence sequence) or the so-called process nets [54] to represent scenarios, as they abstract implementation details and can express concurrency without necessarily determine precise causality between events.

Given a Petri net and an LPO, the proposed approach constructs a process net that respects the concurrency relations formulated by the LPO. Therefore, it is robust to duplicate transitions. However, the approach does not consider Petri nets with invisible transitions. Moreover, since the goal of the approach is to give a yes or no answer to whether a scenario can be executed in a Petri net, it does not provide any further diagnostics when the net can only execute parts of the scenario.

3.2.9 Fuzzy Model Replay

Replay techniques are also needed to make an animation on process control-flow. Given a trace and a fuzzy model [75], Günther [74] proposes a heuristic approach to map the occurrence of events in the trace to nodes of fuzzy models in order to animate the way process were performed.

Fuzzy models [75] are *descriptive* models, i.e., their main purpose is for communicating process knowledge. Fuzzy models have relaxed semantics that allow ambiguous interpretation of the way process is performed. A fuzzy model may contain two types of nodes: primitive nodes and cluster nodes. A primitive node is labeled with an activity, while a cluster node aggregates some activities. An activity can only be involved in at most one node. Thus, fuzzy models do not allow for multiple nodes with the same label. Moreover, all nodes in a fuzzy model are associated to some activities. Arcs in a fuzzy model represent relaxed precedence relations between activities, e.g., if there is an arc between primitive nodes a and b in a fuzzy model, the occurrence of a *may be followed* by b.

An activity can only be mapped to at most one node in a fuzzy model. Therefore, mapping events in a trace to nodes of a fuzzy model is trivial. Moreover, fuzzy models also allow occurrences of activities in ways that are not expressed by their arcs. The semantics of a node, e.g., an AND-split or XOR-split, is derived heuristically by comparing the set of successors of currently "active" nodes and compare it against the successor of events within a look ahead window whose size is predetermined. For example, suppose that the size of look ahead window is 4, if an event e in a trace is mapped to a primitive node n of a fuzzy model and events of all successors of are included in the next four positions in the trace, the semantics of n is AND-split.

Such a straightforward method to map events to process models implies that the approach to replay traces on fuzzy model is highly scalable. However, the relaxed semantics of fuzzy models do not help to identify possible causes of deviations. Similar problem also occur in general for replay approaches with relaxed semantics, such as replaying event logs on SPDs [2] and replaying event logs on process maps [19].

3.2.10 Log Replay on Declarative Models

All techniques mentioned in Section 3.2.1 to Section 3.2.9 require an *imperative* process model, i.e., a model that describes *how the process should "behave"*. An imperative model of a process explicitly shows all possible execution of the process. Consequently, such a model typically lacks flexibility and may be over-specific [142]. This motivates the emergence of *declarative process modeling languages*. Instead of expressing the allowed behavior of a process in a procedural manner, a declarative model expresses only the disallowed behavior. Examples of such modelling languages are Declare [124,181], DCR Graphs [82], and SCIFF [111]. For the sake of completeness, we also investigate replay approaches on declarative models.

Given a trace and a declarative process model in Declare formalism, de Leoni et al. [43] propose an approach to "replay" the log on the model. The replay result does not only show deviations between the trace and the model, but also diagnostics on the deviations, i.e., activities that must be executed according to the model but do not occur in the trace and vice versa. The approach uses the $\mathbb{A}^\star$ algorithm to find the minimum set of deviations in the trace.

The use of purely declarative approaches, however, is still limited in practice [129, 201]. Declarative models are regarded as well suited for highly volatile environments, but the problems in understanding and maintaining them limit their use [77]. Based on empirical experiments that involve BPM practitioners, Reijers et al. [142] show that a purely declarative approach is less attractive than a hybrid approach that combines the imperative and declarative aspects. Also practitioners acknowledge that some processes can be modeled most naturally using the imperative approach, while others would fit better with the declarative approach [142].

3.3 Comparison of Existing Approaches

Table 3.1 shows the summary of a high-level comparison of existing approaches to relate observed and modeled behavior. Given the focus of this thesis, we only consider the approaches that work on imperative process models. As shown in the table, there is no approach that fully supports all criteria described in Section 3.1. Many approaches support duplicate and invisible transitions, but only half of them fully support both types of transitions. All approaches that fully support duplicate transitions are based on state-space analysis. They try all possible solutions before deciding which one is the best to use. Some approaches that partially support for duplicate transitions, e.g., [151]

Table 3.1: Summary of comparison between approaches to relate observed to modeled behavior

Features	Token-based replay [151]	Replay with Artificial Negative Events [68]	Hidden Markov Model Conformance Checking [152]	Posteriori Data Purpose Control Analysis [114]	Measuring Privacy Compliance [16]	Understanding the Behavior of Agents in Business Processes [63]	Comparing Event Streams with Model Streams [39]	Checking Executability of Scenario in Specification [88]	Fuzzy Model Replay [74]
Duplicate transitions	+/−	+/−	+	+	+	+/−	+/−	+	−
Invisible transitions	+/−	+/−	−	+	+	+/−	+/−	−	−
Complex patterns	+/−	+/−	−	+	+/−	+/−	+/−	−	+/−
Loops	+	+	+	+	−	+	+	+	+/−
Deviations	+	+	−	−	+	−	+	−	−
Strict Semantics	+	+	+	+	+	+	+	+	−
Diagnostics	+/−	+/−	−	−	+	−	+	−	−
Scalable	+/−	+	−	+/−	−	−	−	+	+

(+) : fully supported, (+/–) : partially supported, (–) : not supported

and [68], resort to heuristics that may yield misleading results. Similarly, only state-space based approaches can deal with invisible transitions. Here, we see the importance of global analysis to deal with duplicate and invisible transitions. Notice that although the approach to compare event streams with model streams of Cook et al. [39] is based on state-space analysis, it does not fully support either duplicate or invisible transitions. To deal with high computational complexity, a heuristics estimation function that does not guarantee that the solution with the minimum deviation is used in [39]. Without such function, the approach would fully support both types of transitions but is not scalable.

Other than the posteriori data purpose control analysis approach [114], there is no approach that fully supports complex control-flow patterns. Approaches that do not guarantee results with minimum deviations (i.e., [63, 68, 74, 151]) may mistakenly choose wrong semantics that yield misleading result. Both the HMM-based approach [152] and checking executability of scenarios [100] explicitly put models with invisible transitions out-of-scope, hence limit their support to models with complex control-flow patterns. Interestingly, some state-space based analysis techniques (i.e., [16, 39]) also do not fully support complex patterns because they sacrifice guarantee for results with minimum deviations to reduce computation complexity. Only the approach in [114] ignores such computation complexity problems resulting in a poor scalability.

From all approaches, only the privacy compliance measurement in [16] does not support loops, while the heuristics of [74] may mistakenly not identify any loop if a small look-ahead value is used. Many approaches, i.e. [63, 100, 114, 152] are created to simply provide binary answers to whether observed behavior is allowed according to modeled process. Thus, only some of them are insensitive to deviations. Table 3.1 also shows that all approaches that propose some way to deal with deviations also provide diagnostics. Except the fuzzy model replay [74], all approaches work on process models with strict semantics.

The top three approaches with the least number of not supported features are the

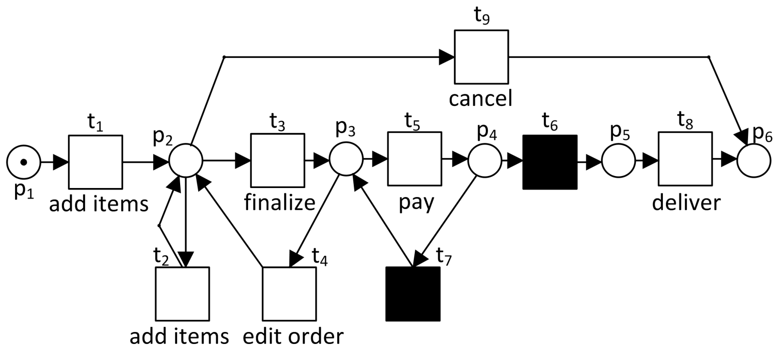


Figure 3.4: An online transaction for an electronic bookstore in Petri net.

token-based replay [151], the replay with artificial negative events [68], and the comparison between event streams and model streams [39]. The heuristics proposed in [68] are not as sophisticated as the ones proposed by [151]. Both of them have inherent problems in dealing with duplicate/invisible transitions, dealing with complex control-flow patterns, and providing useful diagnostics if deviations occur almost in all states of the modeled processes. The comparison between event and model streams [39] does not always guarantee solutions with minimal number of deviations because the proposed use heuristics to deal with its computation complexity in practice.

In this thesis, we propose a robust way to relate observed to modeled behavior using the notion of *alignments*. We formalize alignments in Section 3.4.

3.4 Defining Alignments

Given a process model as a reference, the execution of an activity that does not deviate from the model implies that the activity is allowed by the model in its current state. In the other way around, given a non-deviating execution of the process (e.g., event log), all executed activities can be mimicked by series of actions allowed according to the model. Based on this idea, we define an *alignment* between a recorded process execution and a process model as a pairwise comparison between executed activities in the execution and the activities allowed by the model. Typically, an instance of a process (i.e., case) does not directly influence other instances of the same process. Take for example an insurance claim handling process in an insurance company. The way a claim is handled does not influence the way other claims are handled. They may be competing for the same resources, but this does not directly influence the control-flow. Thus, an alignment is defined per process instance.

We use the Petri net in Figure 3.4 as our running example to explain the concept of alignments. Figure 3.4 shows a Petri net of an online transaction process for an electronic bookstore. A customer can add as many items as he wants to his cart (add items) before finalizing an order (finalize). After the order is finalized, the customer can choose either to pay (pay) or to edit the order (edit order). All payments are performed online, thus there is no guarantee that a payment (pay) is always successful. Success/failure of payments are modeled with two invisible transitions t_6 and t_7 respectively. In case payment fails, the customer can retry another payment or edit the previous order (edit order) and cancel his order (cancel). A successful payment is followed by product delivery (deliver).

	add items	*add items*	*cancel*
$\gamma_1 =$	*add items*	*add items*	*cancel*
	t_1	t_2	t_9

Figure 3.5: An alignment between $\sigma_1 = \langle \mathit{add\ items}, \mathit{add\ items}, \mathit{cancel} \rangle$ and the model in Figure 3.4.

	add items	*cancel*	*finalize*	$\gg$	*finalize*	*pay*	$\gg$	*deliver*
$\gamma_2 =$	*add items*		*finalize*	*edit order*	*finalize*	*pay*		*deliver*
	t_1	$\gg$	t_3	t_4	t_3	t_5	t_6	t_8

Figure 3.6: An alignment between $\sigma_3 = \langle \mathit{add\ items}, \mathit{cancel}, \mathit{finalize}, \mathit{finalize}, \mathit{pay}, \mathit{deliver} \rangle$ and the model in Figure 3.4.

An instance of a process is recorded as a trace in an event log of the process. Given a trace and a Petri net, if the trace perfectly fits the net each activity in the trace can be mimicked by firing a transition in the net. Furthermore, at the end of the trace the final state should have been reached. Take for example a perfectly fitting trace $\sigma_1 = \langle \mathit{add\ items}, \mathit{add\ items}, \mathit{cancel} \rangle$. Figure 3.5 shows an alignment γ_1 between σ_1 and the net in Figure 3.4. The top row of the alignment represents the trace σ_1, while the bottom row represents a complete firing sequence of the Petri net shown in Figure 3.4. All activities in σ_1 are paired with transitions with the same label. We write transition identifier on the bottom row to distinguish transitions with the same label, e.g., t_1 and t_2 are both labeled add items.

Consider a non-fitting trace for the same Petri net $\sigma_2 = \langle \mathit{add\ items}, \mathit{cancel}, \mathit{finalize}, \mathit{finalize}, \mathit{pay}, \mathit{deliver} \rangle$. In this case, not all activities in the trace can be mimicked by firing transitions in the net. Figure 3.6 shows an alignment γ_2 between σ_2 and the net in Figure 3.4. Deviations occur at positions where either a top or bottom row contains the "no move" symbol: $\gg$. For example, the second column in the alignment shows that activity cancel occurs in σ_2 while the net does not allow cancel to occur. The fourth column shows the other way around: transition t_4 (*edit order*) should occur according to the net but it does not occur in the trace. Notice that the occurrence of invisible transition t_6 is also marked as a deviation. Since the execution of invisible transitions is not recorded in event log, they are always shown as a deviation in alignments (event though they are harmless).

Both γ_1 and γ_2 in Figure 3.5 and Figure 3.6 are examples of alignments. Alignments are constructed by pairing activities in traces with transition allowed by process models. Such sequences of pairs are called *movement sequences*. For convenience, in the remainder of this thesis, for any set S, $S^{\gg} = S \cup \{\gg\}$ denotes the union of S with $\{\gg\}$ where $\gg \notin S$. We formalize movement sequences as follows:

Definition 3.4.1 ((Legal) Movement Sequence)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. A *movement sequence* $\gamma \in (A^{\gg} \times T^{\gg})^*$ between σ and N is a sequence of pairs such that

- $\pi_1(\gamma)_{\downarrow A} \leq \sigma$, i.e. its sequence of movements in the trace (ignoring $\gg$) is a prefix of trace σ,
- There exists a complete firing sequence $m_i \xrightarrow{\varrho} m_f, \varrho \in T^*$ such that $\pi_2(\gamma)_{\downarrow T} \leq \varrho$, i.e., its sequence of movements in the model (ignoring $\gg$) is a prefix of a complete firing sequence,

For all tuples $(a, t) \in \gamma$ in a movement sequence, we say that (a, t) is one of the following *movements*:

- *move on log* if $a \in A$ and $t = \gg$,
- *move on model* if $a = \gg$ and $t \in T$,
- *synchronous move* if $a \in A$ and $t \in T$,
- *illegal moves* otherwise.

All moves on log, moves on model, and synchronous moves are considered as *legal moves*. A movement sequence is a *legal movement sequence* if it contains only legal moves.

$\lrcorner$

The definition of legal movement sequence in Definition 3.4.1 allows for a synchronous move (a, t) where the label of transition t is not the same as a. This way, we address situations where an activity in a legal movement needs to be paired with a transition with different label. For example, suppose that the payment activity (pay) shown in Figure 3.4 can be replaced with a transfer money activity (e.g., transfer). Movement $(\text{transfer}, t_5)$ is a legal movement (synchronous move). As another example, suppose that resource names are also a part of activity names in Figure 3.4 and the delivery activity (deliver) can only be approved by a logistic officer, i.e., the label of transition t_8 is (deliver,logistic officer). In an emergency situation, a manager of the company may approve the activity instead of the officer, i.e., activity (deliver,manager) is performed. Hence, Definition 3.4.1 allows for a synchronous move $((\text{deliver,manager}), t_8)$ having a predefined cost.

We use the definition of movement sequences to define an alignment as follows:

Definition 3.4.2 (Alignment)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. An *alignment* $\gamma \in (A^{\gg} \times T^{\gg})^*$ between σ and N is a legal movement sequence such that:

- $\pi_1(\gamma)_{\downarrow A} = \sigma$, i.e. its sequence of movements in the trace (ignoring $\gg$) yields the trace, and
- $m_i \xrightarrow{\pi_2(\gamma)_{\downarrow T}} m_f$, i.e. its sequence of movements in the model (ignoring $\gg$) yields a complete firing sequence of N.

$\Gamma_{\sigma,N}$ is the set of all alignments between a trace σ and a Petri net N. $\lrcorner$

The middle row of γ_1 and γ_2 in Figure 3.5 and Figure 3.6 can be derived using labeling functions. Therefore, the definition of both movement sequences and alignments in Definition 3.4.1 and Definition 3.4.2 does not need to mention transition labels explicitly. Note that alignments require termination of both trace and process model. Thus, no alignment can be constructed for Petri nets whose final marking is not reachable from the initial marking.

For simplicity, in this thesis we only consider easy sound Petri nets with one final marking (see Definition 2.4.4). In practice, a Petri net may have no final marking, e.g., a net that expresses the behavior of a process that "loops" infinitely. In contrast, a Petri net may have multiple final markings reachable from its initial marking. We can easily use the definition of alignments in Definition 3.4.2 to reveal deviations on both types of nets. The idea is given a trace and a Petri net with multiple final markings, we construct a net that (1) has exactly one final marking, and (2) allows for the same set of traces as the original net. For example, we add an extra place p to the original net, and then for each final marking of the original net we add an invisible transition that takes all tokens in the

	add items	*cancel*	*finalize*	*finalize*	*pay*	*deliver*
$\gamma_2' =$	*add items*	*cancel*				
	t_1	t_9	$\gg$	$\gg$	$\gg$	$\gg$

Figure 3.7: Another possible alignment between the same trace and net used to construct the alignment in Figure 3.6, but with more deviating columns.

marking and put a token in the extra place p. It is trivial to see that the constructed net is an easy sound Petri net with one final marking $[p]$. An alignment between the trace and the constructed net yields the deviations between the trace and the original net. A net that expresses an infinite loop behavior can be viewed as a net whose all reachable states are final markings. Thus, given a trace and such a net, we use the same approach as the one used to align traces with Petri nets with multiple final markings. Note that in this thesis, we do not consider process models whose termination states are not reachable, as it indicates that they possibly have modeling issues.

Given a trace and a Petri net, there can be multiple alignments that one can construct. Figure 3.7 shows another alignment between the same trace and process model as the one used to construct alignment γ_2 in Figure 3.6. Note that there are more deviating columns in the alignment shown in Figure 3.7 than the ones shown in Figure 3.6 (4 against 3 deviations). In practice, the likelihood of deviations may differ between activities. For example, one may consider the move on log on activity *pay* to be less likely than move on model on t_6, because nothing is logged for all executions of t_6. To construct a most likely alignment between the trace and the net, we assign a *likelihood cost* to each movement (i.e., synchronous move, move on model, and move on log). We are interested in alignments with the least total likelihood cost according to the assigned likelihood cost function. Such an alignment is called an *optimal alignment*.

Definition 3.4.3 (Optimal alignment)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $lc : A^{\gg} \times T^{\gg} \rightarrow I\!R$ be a likelihood cost function for movements.

We say that $\gamma \in \Gamma_{\sigma,N}$ is an *optimal alignment* between σ and N if and only if for all $\gamma' \in \Gamma_{\sigma,N} : \sum_{(a,t)\in\gamma} lc((a,t)) \leq \sum_{(a',t')\in\gamma'} lc((a',t'))$. $\Gamma^o_{\sigma,N,lc}$ is the set of all optimal alignments between σ and N with respect to likelihood cost function lc.

⌟

Given a trace, an easy sound Petri net, and a likelihood cost function, there exists an upperbound value of the total cost of the of optimal alignments between the trace and the net. In the worst case, an optimal alignment between the trace and the net consists of the moves on log of all activities in the trace and the moves on model of all transitions in a complete firing sequence of the net with the least total of likelihood cost.

Definition 3.4.4 (Likelihood cost limit)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $lc : A^{\gg} \times T^{\gg} \rightarrow I\!R$ be a likelihood cost function for movements. The *likelihood cost limit of optimal alignments* $lim(\sigma, N, lc) \in I\!R$ between σ and N with respect to cost function lc is

$$lim(\sigma, N, lc) = \sum_{a\in\sigma} lc((a, \gg)) + \min(\{\sum_{t\in\varrho} lc((\gg, t)) \mid \varrho \in T^* \wedge m_i \xrightarrow{\varrho} m_f\})$$

i.e., the sum of (the total likelihood cost of all moves on log for all activities in σ) and (the minimum total likelihood cost of an alignment between an empty trace and N). ⌟

Proposition 3.4.5 (Likelihood cost limit is an upperbound)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $lc : A^{\gg} \times T^{\gg} \rightarrow I\!R$ be a likelihood cost function for movements. For all optimal alignments $\gamma \in \Gamma^{o}_{\sigma,N,lc} : \sum_{(x,y)\in\gamma} lc((x,y)) \leq lim(\sigma, N, lc)$. ⌟

Proof. We prove this proposition by showing that there exists an alignment between the trace and the net with total likelihood cost equal to $lim(\sigma, N, lc)$. Let $\varrho \in T^*$ be a complete firing sequence that yields a limit total cost such that $m_i \xrightarrow{\varrho} m_f$ and $\sum_{t\in\varrho} lc((\gg, t)) = \min(\{\sum_{t\in\varrho'} lc((\gg, t)) \mid \varrho' \in T^* \wedge m_i \xrightarrow{\varrho'} m_f\})$. Let $\gamma' = \langle(\sigma[1], \gg), \dots, (\sigma[|\sigma|], \gg)\rangle \cdot \langle(\gg, \varrho[1]), \dots, (\gg, \varrho[|\varrho|])\rangle$ be a movement sequence. It is trivial to see that $\gamma' \in \Gamma_{\sigma,N}$. □

The likelihood cost function provides a certain level of flexibility to determine the desired optimal alignment between a given trace and model. In this thesis, we are mostly interested in alignments to identify deviations between the trace and the model. Thus, we do not penalize synchronous moves and moves on model involving an invisible transition (silent step). In case of a synchronous move, both log and model agree. In case of a silent step, an unlogged transition is fired. We define a *standard likelihood cost function* that assigns zero cost to all synchronous moves of activities and transitions with the same label, as well as to all moves on model of invisible transitions. Furthermore, the function assigns cost 1 to all moves on log/moves on model of normal (not invisible) transitions. The function assigns cost $+\infty$ to all synchronous moves whose transitions have different labels than their activities. In the remainder of the thesis, we use the standard likelihood cost function unless indicated otherwise.

Definition 3.4.6 (Standard likelihood cost function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. The *standard likelihood cost function* $lc : A^{\gg} \times T^{\gg} \rightarrow I\!R$ is the function that maps all movements to real values, such that for all $(x, y) \in A^{\gg} \times T^{\gg}$:

- $lc((x, y)) = 0$ if either $x \in A, y \in T$, and $x = \alpha(y)$, or $x = \gg, y \in T$, and $\alpha(y) = \tau$,
- $lc((x, y)) = +\infty$ if either $x \in A, y \in T$, and $x \neq \alpha(y)$, or $x = y = \gg$, and
- $lc((x, y)) = 1$ otherwise.

⌟

Take for example alignments γ_2 in Figure 3.6 and γ'_2 in Figure 3.7. Using the standard likelihood cost function, the total costs of alignment γ_2 and γ'_2 are 2 and 4 respectively. Moreover, there is no other alignment with less total costs than γ_2 for the trace and the model in Figure 3.4. Hence, γ_2 is an *optimal alignment*.

The notion of optimal alignments provides a robust way to relate occurrence of logged events to process models. Both duplicate and invisible transitions are explicitly handled by optimal alignments, e.g., transition t_2 in the alignment in Figure 3.5 and transition t_6 in the alignment in Figure 3.6. The requirement for having the least possible total likelihood costs also makes the optimal alignment concept robust to loops and complex control flow patterns, because it does not allow unnecessary behavior unless it is the one that minimizes the total cost of likelihood. Furthermore, the alignment explicitly shows where and why deviations occur if there exists any (see the moves on model and the moves on log in Figure 3.6). Remark that for simplicity, in this section all definitions use a process model in terms of Petri nets without reset/inhibitor arcs. However, all definitions can be easily extended to nets with reset/inhibitor arcs or any other executable process models with a well-defined initial (m_i) and final (m_f) state.

Given a trace, a Petri net, and a likelihood cost function for movements, an alignment between the trace and the net with the highest likelihood may not necessarily be an optimal alignment, i.e., it may not always be an alignment with the least total likelihood costs of movements. Therefore, we define an "oracle" function that maps traces to alignments with their probabilities. The higher the probability of an alignment of a trace, the more likely the alignment is the "best" representation of the trace. In the remainder of this thesis, $\mathbb{P} = \{i \in \mathbb{R} \mid 0 \leq i \leq 1\}$ denotes all real values between 0 and 1.

Definition 3.4.7 (Oracle function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let N be an easy sound Petri net over A. An *oracle function* $orc_{L,N} : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow \mathbb{P})$ of L and N is a function that maps traces to alignments with probabilities, such that for all $\sigma \in L$:

- $\forall_{\gamma \in (A^{\gg} \times T^{\gg})^*} orc_{L,N}(\sigma)(\gamma) > 0 \implies \gamma \in \Gamma_{\sigma,N}$, i.e., traces with non-zero occurrences are mapped to alignments between σ and N,
- $\sum_{\gamma \in \Gamma_{\sigma,N}} orc_{L,N}(\sigma)(\gamma) = 1$, i.e., the sum of the probabilities of all alignments of a trace is equal to 1.

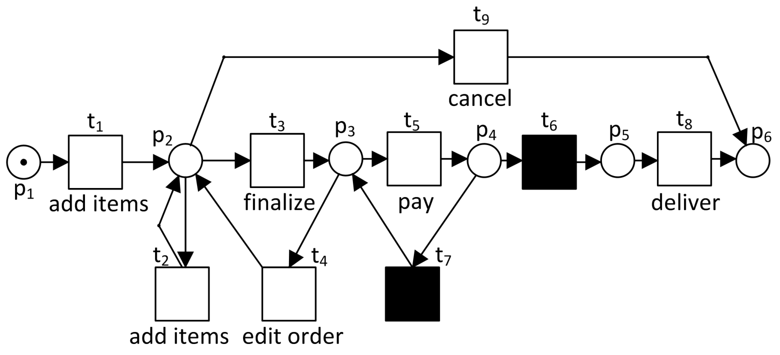
The subscripts L and N in $orc_{L,N}$ can be omitted if the context is clear. ⌟

An oracle function gives the probabilities of all possible alignments between a given trace and Petri net. Take for example trace $\sigma_3 = \langle \textit{add items}, \textit{cancel}, \textit{finalize}, \textit{finalize}, \textit{pay}, \textit{deliver} \rangle$ and the Petri net N in Figure 3.8. The bottom part of Figure 3.8 shows two alignments between them as shown previously in Figure 3.6 and Figure 3.7. Using the standard likelihood cost function, γ_2 is an optimal alignment between σ_3 and N.

Let $L = [\sigma_3]$ be a log of one trace σ_3. Suppose that alignment γ_2 has 100% probability to represent the relation between trace σ_3 and net N. Based on this information, we can make an oracle function that returns the probability of 1 to alignment γ_2 and 0 to other alignments, given trace σ_3 and net N. Thus, $orc^1(\sigma_3)(\gamma_2) = 1$, $orc^1(\sigma_3)(\gamma_2') = 0$, and $orc^1(\sigma_3)(\gamma) = 0$ for all other alignments $\gamma \in \Gamma_{\sigma_3,N}$ between σ_3 and N. As another example, if the probability of γ_2 is 70% and the probability of γ_2' is 30% then we can also define an oracle function orc^2 such that $orc^2(\sigma_3)(\gamma_2) = 0.7$, $orc^2(\sigma_3)(\gamma_2') = 0.3$, and $orc^2(\sigma_3)(\gamma) = 0$ for all other alignments $\gamma \in \Gamma_{\sigma_3,N}$ between σ_3 and model N.

We define three special oracle functions: the *standard oracle function*, the *basic oracle function*, and the *optimal oracle function*. Given a trace and a Petri net, a standard oracle function yields a set of non-empty alignments between the trace and the net, where each alignment has the same probability. A basic oracle function yields only a single alignment between the trace and the net, i.e., it assigns a probability of 1 to an alignment and 0 to all other alignments between the trace and the net. An optimal oracle function yields a set of non-empty optimal alignments between the trace and the net with respect to a given cost function.

Definition 3.4.8 (Standard, basic, and optimal oracle function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let N be an easy sound Petri net over A, and let $orc_{L,N} : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow \mathbb{P})$ be an oracle function of L and N.



$\gamma_2 =$

add items	*cancel*	*finalize*	$\gg$	*finalize*	*pay*	$\gg$	*deliver*
add items		*finalize*	*edit order*	*finalize*	*pay*		*deliver*
t_1	$\gg$	t_3	t_4	t_3	t_5	t_6	t_8

$\gamma'_2 =$

add items	*cancel*	*finalize*	*finalize*	*pay*	*deliver*
add items	*cancel*				
t_1	t_9	$\gg$	$\gg$	$\gg$	$\gg$

Figure 3.8: Top: an online transaction for an electronic bookstore shown in Figure 3.4, **Bottom:** Two possible alignments between $\sigma_3 = \langle \mathit{add\ items}, \mathit{cancel}, \mathit{finalize}, \mathit{finalize}, \mathit{pay}, \mathit{deliver} \rangle$ and the net.

$orc_{L,N}$ is a *standard oracle function* if for all $\sigma \in L, \gamma, \gamma' \in \Gamma_{\sigma,N} : orc_{L,N}(\sigma)(\gamma) > 0 \wedge orc_{L,N}(\sigma)(\gamma') > 0 \implies orc_{L,N}(\sigma)(\gamma) = orc_{L,N}(\sigma)(\gamma')$, i.e., all alignments of a trace with a probability higher than 0 have the same probability value.

$orc_{L,N}$ is a *basic oracle function* if for all $\sigma \in L, \gamma \in \Gamma_{\sigma,N} : orc_{L,N}(\sigma)(\gamma) > 0 \implies orc_{L,N}(\sigma)(\gamma) = 1$, i.e., there is only one alignment per unique trace.

Let $lc : (A^{\gg} \times T^{\gg})^* \to \mathbb{R}$ be a function that maps all movements to real values. $orc_{L,N}$ is an *optimal oracle function* with respect to lc if for all $\sigma \in L, \gamma \in \Gamma_{\sigma,N} : orc^{lc}_{L,N}(\sigma)(\gamma) > 0 \implies \gamma \in \Gamma^{o}_{L,N,lc}$, i.e., each trace is mapped to a set of optimal alignments with respect to cost function lc. $orc^{lc}_{L,N}$ denotes an optimal oracle function with respect to lc.

$\lrcorner$

The previously mentioned orc^1 is an example of a basic, standard, and optimal oracle function with respect to the standard likelihood cost. These three types of oracle functions can be used to support different types of analysis. We show the application of such functions in Part III of this thesis.

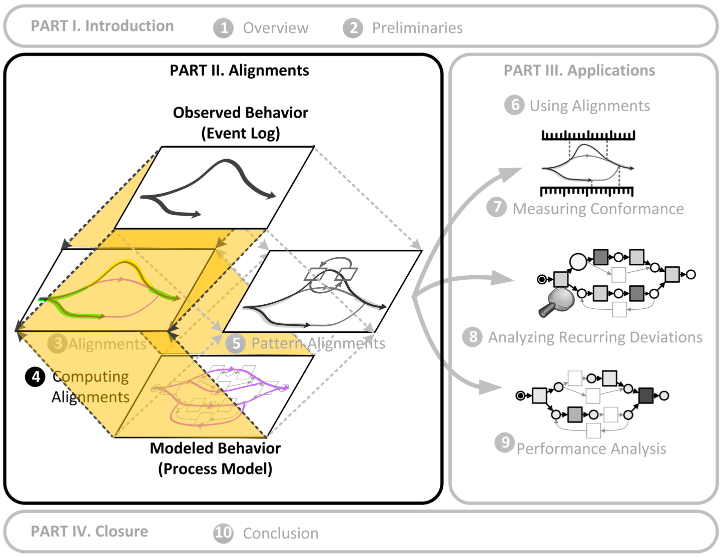
3.5 Conclusion

Relating the occurrences of logged events with transitions of Petri nets (i.e., replay) is far from trivial. Many approaches have been proposed in literature, but none of them satisfies all requirements identified. In this section, we defined alignments and optimal alignments as robust concepts to relate logged events with process models. We showed that optimal alignments can deal with peculiarities of process models such as duplicate/invisible transitions (i.e., in Petri net terms), complex control-flow, and loops. In situations where deviations occur, alignments explicitly show where the deviations are and hence provide diagnostics information that can be further exploited. Furthermore, the flexibility to assign likelihood cost functions also provides a way to consider the severity of deviations in constructing alignments. This flexibility can be further exploited to obtain alignments that are particularly suitable for a specific type of analysis.

In this chapter, we also described the notion of oracle functions without discussing the details on how the functions can be implemented. In Chapter 4, we describe some approaches to construct optimal alignments efficiently, given a trace and a Petri net. These approaches can be used to realize the oracle functions introduced in this chapter.

Chapter 4

Computing Alignments



4.1 Introduction

Given a trace and a process model, constructing an optimal alignment between them requires the computation of a sequence of activities that is allowed by the model and is closest to the trace. Such a computation can be very expensive, especially in cases where the model allows for an infinite number of traces. A brute force method of listing all possible behaviors of the model and choose the one that is closest to the trace is clearly not feasible if the model allows for infinitely many traces. A systematic approach to construct an optimal alignment that does not require listing all behaviors allowed by the model is needed.

Given a trace and a Petri net, in this section we provide an approach based on optimization techniques to compute an optimal alignment between them efficiently. In Section 4.2, we provide an overview of related work. In Section 4.3, we show how the problem of computing optimal alignments can be modeled as a problem of finding a firing sequence of Petri nets. Section 4.4 provides a detailed explanation of a memory-efficient approach to compute an optimal alignment between the trace and the net. Section 4.5 introduces the notion of prefix alignments as an alternative to optimal alignments to identify deviations between the trace and the model if the trace is known to be incomplete. Section 4.6 shows some extensions to the approach explained in Section 4.4 and Section 4.5 to obtain more than one (prefix) alignments and representatives of all optimal (prefix) alignments from a given trace and a given Petri net. Experiment results for the proposed approaches are given in Section 4.7. Section 4.8 concludes this chapter.

4.2 Related Work

Given an event stream and a process model, Cook et al. [39] proposed a state-space analysis approach to find all deviations between the stream and the streams allowed by the model. To identify deviations between them, the approach builds a tree of states iteratively where each state consists of a pair of event and model streams. To prevent a state-space explosion, the tree is constructed using a best-first search strategy where states that are most likely to lead to solutions are explored first before others. However, this approach has potential memory problems dealing with models that exhibit parallelism. New states are appended as leaves of the state space tree. Therefore, the same state may appear in more than one branch of the tree. This redundancy problem may lead to poor memory-efficiency. To solve this problem, some heuristics are suggested to prune the state space tree. However, applying the heuristics removes the guarantee that the minimum number of deviations will be found.

The problem of finding optimal alignments also has some similarities to the problem of finding edit distance between two strings. Given two finite strings, Levenshtein [98] defines the distance between them as the minimum number of insertion, deletion, and substitution operations to transform one of them into the other and vice versa. The minimum edit distance between two finite strings can be efficiently computed using dynamic programming approaches such as the Needleman-Wunsch algorithm [121]. Given a trace and a process model, the locations of moves on model and moves on log in an optimal alignment between them show explicitly where activities need to be added/removed to transform the trace into a sequence of activities allowed by the model. However, there is a substantial difference between constructing optimal alignments and computing minimum string edit distances. Process models may allow (infinitely) many behaviors, while string edit distance computations require strings of finite length. Even if the number of traces allowed by the process model is finite, choosing one of the allowed traces that is most similar to the given trace can be computationally challenging as shown by Banescu et al. [16]. To overcome the complexity, Banescu et al. [16] suggested some heuristics and an approach based on dynamic programming. However, the obtained "alignments" from this approach are not guaranteed to be optimal.

The problem of constructing optimal alignments can be viewed as an optimization problem. Given a set of constraints, an objective function, and a set of decision variables, there exists many optimization approaches to identify the values of the variables that maximize/minimize the objective function without violating any of the constraints [34]. Given a trace, a process model, and a likelihood cost function, constructing an optimal alignment between the trace and the model can be viewed as the problem of constructing a sequence of pairs of movements with a set of constraints (see Definition 3.4.3)

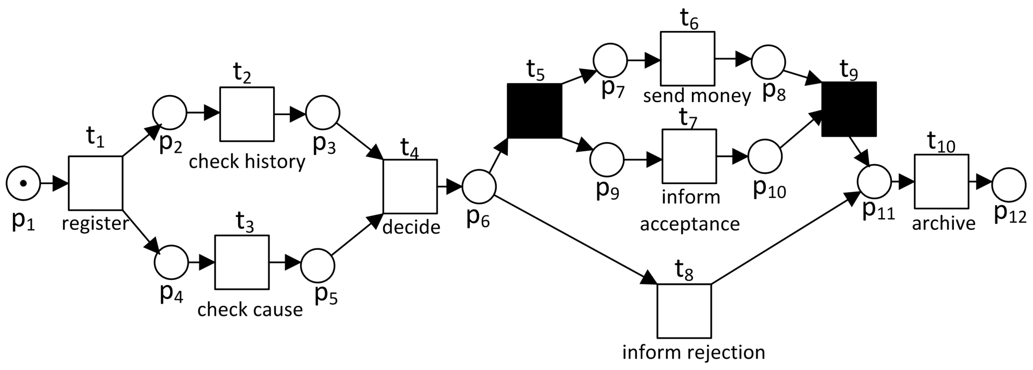


Figure 4.1: A process of handling insurance claim in an insurance company.

and an objective function defined as the total cost of movements. The goal of the optimization is to minimize the objective function. Thus, optimization approaches such as genetic algorithms [109] and linear programming approaches [197] are applicable to solve such a problem. However, optimization problems require constraints in form of equality/inequality algebraic equations, while the constraints of optimal alignments also includes the possible ordering of activities as defined by process models. Given a process model, there can be infinitely many possible ordering of activities (i.e., many allowed behavior). Such an infinite behavior may not be easily represented in forms of algebraic equations.

The construction of optimal alignments from traces and process models resembles the modeling of concurrent processes with possible synchronization. Given two concurrent processes represented in Petri nets where some pairs of transitions need to be synchronized, Winskel [208] defines the product of the two nets to show all possible synchronization between the two nets explicitly. Such an explicit representation enables the behavioral analysis of synchronous systems using existing techniques. To construct an optimal alignment between a trace and a Petri net, activities in the trace need to be "synchronized" with transitions in the net. However, analysis approaches based on the product of two Petri nets typically do not take into account cost functions, while optimal alignments are defined with respect to likelihood cost functions.

4.3 Modeling Alignment Problems

Given a trace and an easy sound Petri net, an optimal alignment between them is constructed by performing a sequence of movements that changes the state of the trace, state of the net, or both. A synchronous move is a movement that changes both the state of the trace and the state of the net, while other movements (i.e., moves on log, moves on model) only change the state of either the net or the trace. Such a relation between the trace and the net resembles the relation between two synchronized concurrent processes. We explicitly model the change of states in both the trace and the net as a synchronization problem between two Petri nets.

We use the easy sound Petri net in Figure 4.1 as a running example. Figure 4.1 shows a process of handling insurance claims in an insurance company. An instance of the process starts from the moment an officer register a claim from a claimant (register). Then, both the claimant history (check history) and causes of the claim (check causes) are

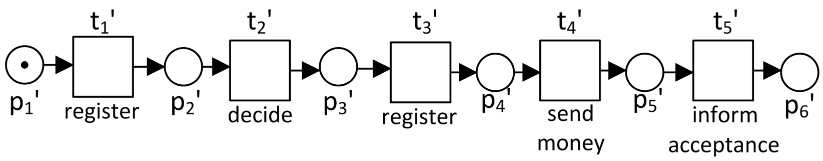


Figure 4.2: The event net of trace $\sigma_1 = \langle register, decide, register, send\ money, inform\ acceptance\rangle$.

checked in an arbitrary order. A decision is made by a manager (decide) based on the checking results. If the claim is accepted, the claimant is informed (inform acceptance) and the claimed amount of money is send (send money). If the claim is rejected, the reason of rejection is informed immediately to the claimant (inform rejection). The process ends after the claim is archived by an insurance officer (archive).

Suppose that we want to find an optimal alignment between a trace $\sigma_1 = \langle register, decide, register, send\ money, inform\ acceptance\rangle$ and the net in Figure 4.1 using the standard likelihood cost function. We first construct the *event net* [6] of the trace where all possible states of the trace are captured explicitly by the net's marking. The event net of a trace is a Petri net with a linear structure, such that each transition in the net represents a unique activity occurrence in the trace. Figure 4.2 shows the event net of σ_1.

Formally, we define event net of a simple trace as follows:

Definition 4.3.1 (Event net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace of length n over A. The *event net* of σ is a Petri net $N = (P, T, F, \alpha, m_i, m_f)$, where

- $P = \{p_j \mid 1 \leq j \leq n+1\}$,
- $T = \{t_j \mid 1 \leq j \leq n\}$,
- $F : (P \times T) \cup (T \times P) \to I\!N$, such that
  - $F(p_j, t_j) = 1$, for all $1 \leq j \leq n, p_j \in P, t_j \in T$,
  - $F(t_j, p_{j+1}) = 1$, for all $1 \leq j \leq n, p_j \in P, t_j \in T$, and
  - $F(x, y) = 0$ otherwise.
- $\alpha : T \to A$ is a function mapping transitions to activities such that for all $1 \leq j \leq n, \alpha(t_j) = \sigma[j]$,
- $m_i = [p_1]$ is the initial marking, and
- $m_f = [p_{|P|}]$ is the final marking.

⌟

To prevent ambiguity between event nets and easy sound Petri nets that represent process models, from this section on we use the term *process nets* to refer to the latter. Notice that all event nets are sound workflow nets (see Definition 2.4.7).

Having modeled traces as event nets, we explicitly model all possible movements by taking the *product* of two Petri nets: event nets and process nets. The product of two Petri nets is the union of both nets with extra *synchronous transitions* that are constructed by pairing transitions in one net with transitions in the other net that have the same label [208]. This way, possible synchronous moves are modeled by synchronous transitions, while moves on log and moves on model are modeled as unpaired transitions in the event net and process net respectively.

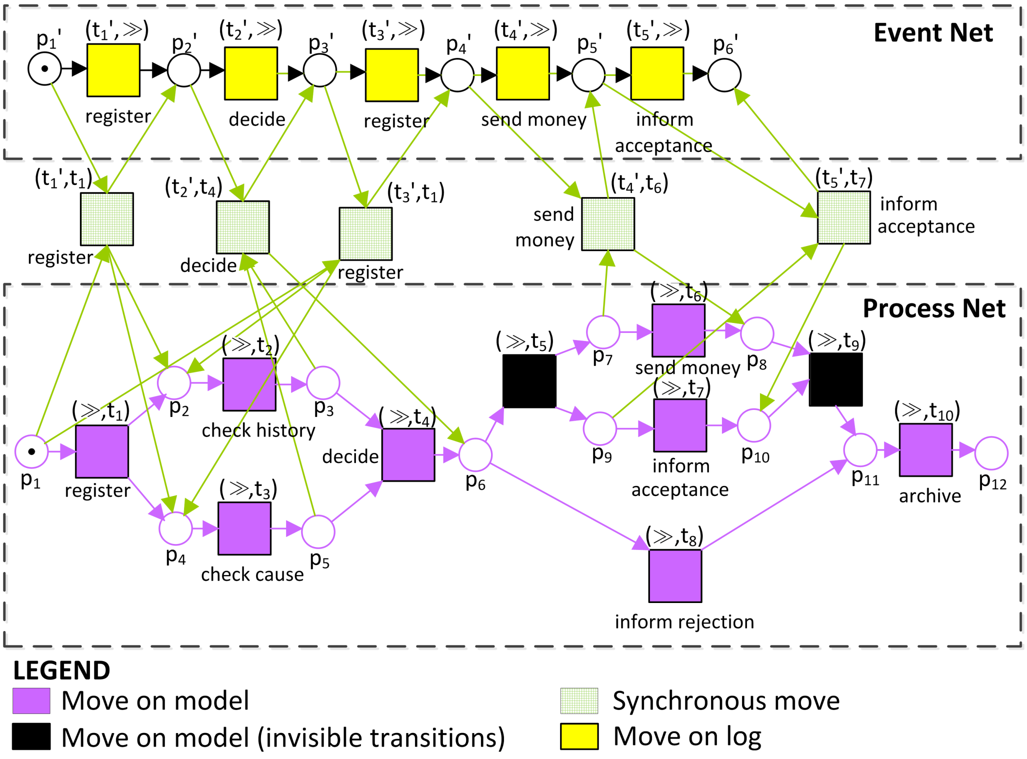


Figure 4.3: The product of the event net of trace $\sigma_1 = \langle$*register*, *decide*, *register*, *send money*, *inform acceptance*$\rangle$ and the model in Figure 4.1.

The product between the event net in Figure 4.2 and the process net in Figure 4.1 is shown in Figure 4.3. The color of transitions and places distinguishes elements of the original nets and the added synchronous transitions. Yellow, purple, black, and green-colored transitions represent moves on log, moves on model (on normal transitions), moves on model (on invisible transitions), and synchronous moves respectively. If a yellow transition is fired, the state of the event net is changed but not the state of the process net. Similarly, firing a purple/black transition only changes the state of the process net. Firing a green transition changes the state of both nets. Notice that all transitions in the net represents movements. For example, transition (t'_1, t_1) represents a synchronous move by doing the first activity *register* in the trace and firing transition t_1 in the process net. As another example, transition $(\gg, t_1)$ represents a move on model by firing the transition t_1 in the process net without moving in the trace.

The product of two Petri nets is formalized as follows [208].

Definition 4.3.2 (Product of two Petri nets)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ and $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be two Petri nets over A. The *product* of N_1 and N_2 is the Petri net $N_3 = N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ where

- $P_3 = P_1 \cup P_2$ is the union set of places,
- $T_3 \subseteq (T_1^{\gg} \times T_2^{\gg})$, such that $T_3 = \{(t_1, \gg) \mid t_1 \in T_1\} \cup \{(\gg, t_2) \mid t_2 \in T_2\} \cup \{(t_1, t_2) \in (T_1 \times T_2) \mid \alpha_1(t_1) = \alpha_2(t_2) \neq \tau\}$ is the set of the original transitions with additional *synchronous transitions*,

- $F_3 : (P_3 \times T_3) \cup (T_3 \times P_3) \to I\!N$ is the arc weight function, such that
  - $F_3(p_1, (t_1, \gg)) = F_1(p_1, t_1)$ if $p_1 \in P_1$ and $t_1 \in T_1$,
  - $F_3((t_1, \gg), p_1) = F_1(t_1, p_1)$ if $p_1 \in P_1$ and $t_1 \in T_1$,
  - $F_3(p_2, (\gg, t_2)) = F_2(p_2, t_2)$ if $p_2 \in P_2$ and $t_2 \in T_2$,
  - $F_3((\gg, t_2), p_2) = F_2(t_2, p_2)$ if $p_2 \in P_2$ and $t_2 \in T_2$,
  - $F_3(p_1, (t_1, t_2)) = F_1(p_1, t_1)$ if $p_1 \in P_1$ and $(t_1, t_2) \in T_3 \cap (T_1 \times T_2)$,
  - $F_3(p_2, (t_1, t_2)) = F_2(p_2, t_2)$ if $p_2 \in P_2$ and $(t_1, t_2) \in T_3 \cap (T_1 \times T_2)$,
  - $F_3((t_1, t_2), p_1) = F_1(t_1, p_1)$ if $p_1 \in P_1$ and $(t_1, t_2) \in T_3 \cap (T_1 \times T_2)$,
  - $F_3((t_1, t_2), p_2) = F_2(t_2, p_2)$ if $p_2 \in P_2$ and $(t_1, t_2) \in T_3 \cap (T_1 \times T_2)$,
  - otherwise $F_3(x, y) = 0$.
- $\alpha_3 : T_3 \to A^\tau$ is the mapping from transitions to activities, such that for all $(t_1, t_2) \in T_3$, $\alpha_3((t_1, t_2)) = \alpha(t_1)$ if $t_2 = \gg$, $\alpha_3((t_1, t_2)) = \alpha_2(t_2)$ if $t_1 = \gg$, and $\alpha_3((t_1, t_2)) = \alpha_1(t_1)$ otherwise,
- $m_{i,3} = m_{i,1} \uplus m_{i,2}$ is the initial marking,
- $m_{f,3} = m_{f,1} \uplus m_{f,2}$ is the final marking.

$\lrcorner$

We define a reverse function that maps transitions in the product of two Petri nets to movements as follows:

Definition 4.3.3 (Reverse function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be a net over A where Rng$(\alpha_1) \subseteq A$ (i.e., there is no transition mapped to τ) and let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be a Petri net over A, and let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2.

$rev_{(N_1 \otimes N_2)} : T_3 \to A^{\gg} \times T^{\gg}$ is the *reverse function* of $N_1 \otimes N_2$ that maps transitions in T_3 to movements, such that for all $(t_1, t_2) \in T_3$:

- $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (\alpha_1(t_1), \gg)$ if $t_2 = \gg$, i.e., all transitions derived from only T_1 are mapped to moves on log,
- $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (\gg, t_2)$ if $t_1 = \gg$, i.e., transitions derived from only transitions of T_2 are mapped to moves on model, and
- $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (\alpha_1(t_1), t_2)$ if $\alpha_1(t_1) \neq \gg$ and $t_2 \neq \gg$, i.e., synchronous transitions are mapped to synchronous moves.

$\lrcorner$

We use the reverse function and the theory of marking reachability in product of two Petri nets in [208] to prove that complete firing sequences of products of event nets and easy sound process nets yield alignments. We can reformulate the result of [208] as follows:

Theorem 4.3.4 (Reachability of marking in the product of two Petri Nets)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ and $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be two Petri nets over A. Let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2. $m_{i,3} \xrightarrow{\varrho}_{(N_1 \otimes N_2)} m, \varrho \in T_3^*$ if and only if both $m_{i,3\downarrow P_1} \xrightarrow{\pi_1(\varrho)_{\downarrow T_1}}_{N_1} m_{\downarrow P_1}$ and $m_{i,3\downarrow P_2} \xrightarrow{\pi_2(\varrho)_{\downarrow T_2}}_{N_2} m_{\downarrow P_2}$ hold, i.e., the projection of the marking to each original net is also reachable from its initial marking.

Proof. Refer to [208]. □

┘

Using this result we prove that firing sequences of such product yields movement sequences, and therefore yields alignments:

Theorem 4.3.5 (Firing sequences define movement sequences)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be its event net. Let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be an easy sound process net over A, and let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2.

For all firing sequences $m_{i,3} \xrightarrow{\varrho}_{(N_1 \otimes N_2)} m, \varrho \in T_3^*$, $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence[1]. Moreover, if $m_{i,3} \xrightarrow{\varrho}_{(N_1 \otimes N_2)} m_{f,3}$, then $rev_{(N_1 \otimes N_2)}(\varrho)$ is an alignment. ┘

Proof. We prove the first part of this lemma by induction. If $\varrho = \langle\rangle$, then $rev_{(N_1 \otimes N_2)}(\varrho) = \langle\rangle$ is an alignment. Assume that $\varrho = \varrho' \cdot (t_1, t_2)$ and that $rev_{(N_1 \otimes N_2)}(\varrho')$ is a movement sequence. There exist markings m_1 and m_2 where $m_{i,3} \xrightarrow{\varrho'}_{(N_1 \otimes N_2)} m_1 \xrightarrow{(t_1,t_2)}_{(N_1 \otimes N_2)} m_2$. By definition, Rng$(\alpha_1) \subseteq A$. We show that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence by considering all three cases:

- Assume $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (a, \gg), a \in A$ (move on log). By definition, Dom$(\alpha_1) = T_1$. Thus, there exists a transition $t_1 \in T_1$ where $\alpha_1(t_1) = \sigma[1 + |\pi_1(\varrho')_{\downarrow T_1}|] = a$ and $m_{i,3_{\downarrow P_1}} \xrightarrow{\pi_1(\varrho')_{\downarrow T_1}}_{N_1} m_{1\downarrow P_1} \xrightarrow{t_1}_{N_1} m_{2\downarrow P_1}$. Furthermore, $m_{i,3_{\downarrow P_2}} \xrightarrow{\pi_2(\varrho)_{\downarrow T_2}}_{N_2} m_{1\downarrow P_2}$. It is easy to see that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence,
- Assume $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (\gg, t_2), t_2 \in T_2$ (move on model). We know that $m_{i,3_{\downarrow P_1}} \xrightarrow{\pi_1(\varrho')_{\downarrow T_1}}_{N_1} m_{1\downarrow P_1}$ and $m_{i,3_{\downarrow P_2}} \xrightarrow{\pi_2(\varrho')_{\downarrow T_2}}_{N_2} m_{1\downarrow P_2} \xrightarrow{t_2}_{N_2} m_{2\downarrow P_2}$ (see Theorem 4.3.4). It is easy to see that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence,
- Assume $rev_{(N_1 \otimes N_2)}((t_1, t_2)) = (a, t_2), a \in A, t_2 \in T_2$ (synchronous move). We know that $m_{i,3_{\downarrow P_2}} \xrightarrow{\pi_2(\varrho')_{\downarrow T_2}}_{N_2} m_{1\downarrow P_2} \xrightarrow{t_2}_{N_2} m_{2\downarrow P_2}$. By definition, $t_1 \in$ Dom(α_1) such that $\alpha(t_1) = \sigma[1 + |\pi_1(\varrho')_{\downarrow T_1}|] = a$ and $m_{i,3_{\downarrow P_1}} \xrightarrow{\pi_1(\varrho')_{\downarrow T_1}}_{N_1} m_{1\downarrow P_1} \xrightarrow{t_1}_{N_1} m_{2\downarrow P_1}$. It is easy to see that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence.

We know that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence. If $m_{i,3} \xrightarrow{\varrho}_{(N_1 \otimes N_2)} m_{f,3}$ then $m_{i,3_{\downarrow P_1}} \xrightarrow{\pi_1(\varrho)_{\downarrow T_1}}_{N_1} m_{f,3_{\downarrow P_1}}$ and $m_{i,3_{\downarrow P_2}} \xrightarrow{\pi_2(\varrho)_{\downarrow T_2}}_{N_2} m_{f,3_{\downarrow P_2}}$. By definition, this implies $m_{i,1} \xrightarrow{\pi_1(\varrho)_{\downarrow T_1}}_{N_1} m_{f,1}$ and $m_{i,2} \xrightarrow{\pi_2(\varrho)_{\downarrow T_2}}_{N_2} m_{f,2}$. We know that the following holds: $\langle \alpha_1(\pi_1(\varrho)_{\downarrow T_1}[1]), \ldots, \alpha_1(\pi_1(\varrho)_{\downarrow T_1}[|\pi_1(\varrho)_{\downarrow T_1}|]) \rangle = \sigma$. Hence, $rev_{(N_1 \otimes N_2)}(\varrho)$ is an alignment. □

Note that the product of an event net and a process net does not model any synchronous move containing an activity and a transition with different label. Hence, such a synchronous move cannot be obtained from the product net. However, Definition 4.3.2

[1]We abuse the reverse function to handle sequences of transitions. Let $\varrho \in T^*$ be a sequence of transitions, $rev_{(N_1 \otimes N_2)}(\varrho) = \langle rev_{(N_1 \otimes N_2)}(\varrho[1]), \ldots, rev_{(N_1 \otimes N_2)}(\varrho[|\varrho|]) \rangle$

can be easily modified to explicitly model such a movement. For simplicity, in the remainder of this chapter we only consider synchronous moves of activities and transitions with the same label.

Given an event net and a process net, Theorem 4.3.5 shows that the problem of constructing alignments can be viewed as the problem of finding some firing sequences in the product of the two nets. Since transitions represent movements, we assign costs to each transition according to the movement it represents. A brute force approach can be used to find firing sequences in the product that yield minimum costs. However, such an approach may perform poorly, especially in situations where the product net allows for many (possibly infinite) firing sequences. In Section 4.4, we propose a state-space exploration strategy to efficiently compute optimal alignments for easy sound process nets.

4.4 Computing an Optimal Alignment

Given a trace and a process model, Section 4.3 shows that the problem of finding an alignment corresponds to finding a firing sequence in the product of two Petri nets. In this section, we provide an *efficient* method to solve the problem using a state-space exploration approach. Section 4.4.1 shows that the problem of computing a firing sequence with minimum cost can be translated into shortest path problems, for which some efficient solutions exist in literature. Section 4.4.2 provides a state space exploration strategy that exploits well-known Petri net results to further improve the computation efficiency in practice. An extension to the approach to handle reset/inhibitor nets is explained in Section 4.4.3.

4.4.1 Translation to Shortest Path Problems

In this section, we model the problem of computing optimal alignments as a shortest path problem. The behavior of a Petri net is explicitly represented by its labeled transition system, which can also be viewed as a directed graph. Take for example the transition system shown in Figure 4.4. The figure shows the transition system of the product net previously shown in Figure 4.3 between the event net of trace $\langle register, decide, register, send\ money, inform\ acceptance\rangle$ and the process net in Figure 4.1. The red-colored state in the transition system is the final marking of the product net. The color of an edge indicates the type of movements it correspond to (i.e., move on log, move on model (invisible transition), move on model (not invisible transition), or synchronous move).

We assign the distance of each edge in the transition system according to the likelihood cost of movement represented by its label. Suppose that we use the standard likelihood cost function. The distance of the edge from the initial state $[p_1, p'_1]$ to $[p_2, p'_2, p_4]$ in Figure 4.4 is 0, because transition (t'_1, t_1) represents a synchronous move and the cost of synchronous move is 0. As another example, the distance of the edge from the initial state $[p_1, p'_1]$ to state $[p'_1, p_2, p_4]$ is 1, because transition $(\gg, t_1)$ represents a move on model. Since an edge in a transition system corresponds to firing a transition, the total distance of a path in the transition system corresponds to the total cost of firing the sequence of transitions along the path, thus representing the total cost of the movement sequence represented by the firing sequence. For example, the highlighted path in Figure 4.13 (i.e., $\langle([p_1, p'_1], (t'_1, t_1), [p_2, p'_2, p_4]), \ldots, ([p_{11}, p'_6], (\gg, t_{10}), [p_{12}, p'_6])\rangle$) can be transformed back to the movement sequence shown in Figure 4.5 by mapping all edges in the path to

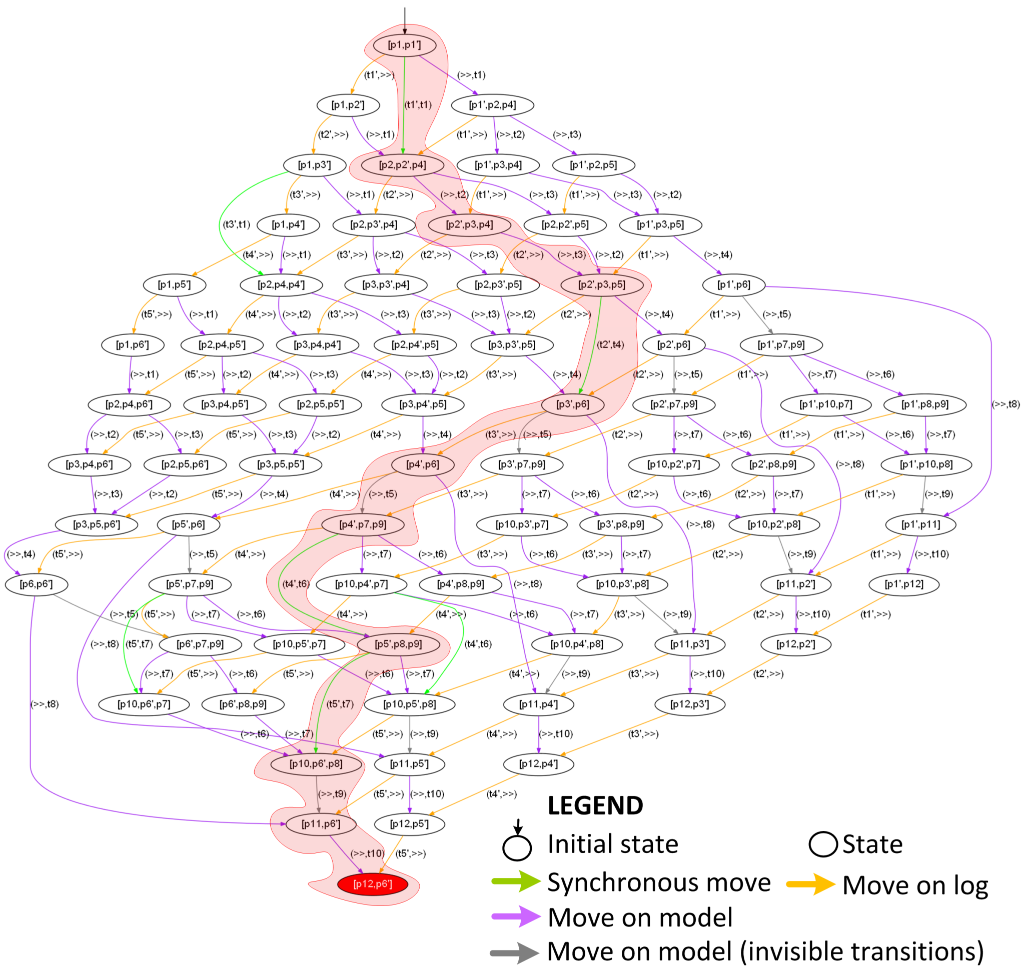


Figure 4.4: The transition system of the product between the event net of trace $\sigma_1 = \langle register, decide, register, send\ money, inform\ acceptance \rangle$ and the model in Figure 4.1. The colored state is the final marking of the product net, i.e., marking $[p_{12}, p'_6]$. The highlighted path from the initial state to state $[p_{12}, p'_6]$ yields the optimal alignment shown in Figure 4.5.

the movements represented by their labels. In this example, the path has a total distance of 4, which is the same as the cost of movement sequence shown in Figure 4.5. Notice that the movement sequence is also an alignment.

Since all paths from the initial state of such a transition system to any state in the same transition system yield movement sequences and the total distance of each path yields the cost of the movement sequence constructed from the path, a shortest path from the initial state to the final state of the transition system yields an optimal alignment between the trace and the net. The alignment shown in Figure 4.5 is an optimal alignment between trace $\langle register, decide, register, send\ money, inform\ acceptance \rangle$ and the process net shown in Figure 4.1, because the highlighted path in Figure 4.4 is a shortest path from the initial state to the final state of the transition system.

We provide a formal proof for all of the arguments mentioned up to this point. First,

register	$\gg$	$\gg$	*decide*	*register*	$\gg$	*send money*	*inform acceptance*	$\gg$	$\gg$
register	*check history*	*check cause*	*decide*			*send money*	*inform acceptance*		*archive*
t_1	t_2	t_3	t_4	$\gg$	t_5	t_6	t_7	t_9	t_{10}

Figure 4.5: The alignment/movement sequence obtained by translating the edges along the highlighted path shown in Figure 4.4 to the represented movements. This sequence is also an optimal alignment between trace $\sigma_1 = \langle \mathit{register}, \mathit{decide}, \mathit{register}, \mathit{send\ money}, \mathit{inform\ acceptance} \rangle$ and the model in Figure 4.1.

we show that there exists a path from the initial state to the final state of any net that is the product of an event net and an easy sound process net.

Lemma 4.4.1 (Path from the initial state to the final state exists)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ and $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be an event net and an easy sound process net over A respectively. Let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$.

$\Psi_{(S,TR,LB)}(s_i, s_f) \neq \emptyset$, i.e., a path from the initial state s_i to the final state s_f of $TS(N_1 \otimes N_2)$ always exists. ⌟

Proof. From Definition 4.3.1, we know that there exists $\varrho_1 \in T_1^*$, such that $m_{i,1} \xrightarrow{\varrho_1}_{N_1} m_{f,1}$. From Definition 2.4.4, we know that there exists a sequence $\varrho_2 \in T_2^*$ such that $m_{i,2} \xrightarrow{\varrho_2}_{N_2} m_{f,2}$. Since markings $m_{f,1}$ and $m_{f,2}$ are reachable from $m_{i,1}$ and $m_{i,2}$ respectively, there exists $\varrho_3 \in T_3^*$ such that $s_i \xrightarrow{\varrho_3}_{N_3} s_f$ (see Theorem 4.3.4). Hence, $\Psi_{(S,TR,LB)}(s_i, s_f) \neq \emptyset$. □

Intuitively, there are many trivial alignments between a trace and a model. It is possible to first do move on model only and then do move on log only for all activities in the trace, as well as all interleavings between its moves on model and moves on log. Typically, such trivial alignments do not correspond to an optimal alignment. In Theorem 4.3.5, we showed that paths in transition systems yield alignments and such path always exists (see Lemma 4.4.1). Hence, we can show that a shortest path from initial states to final states of such transition systems yields an optimal alignment.

Theorem 4.4.2 (Shortest path yields optimal alignment)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let N_1 be an event net over A, let $N_2 = (P, T, F, \alpha, m_i, m_f)$ be an easy sound process net over A. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$. Let $lc : A^{\gg} \times T^{\gg} \to I\!R$ be a likelihood cost function of movements. Let $dist : TR \to I\!R$ be a distance function, such that for all $tr \in TR$, $dist(tr) = lc(rev_{(N_1 \otimes N_2)}(\pi_2(tr)))$.

For all shortest paths $\varrho \in \Psi_{(S,TR,LB)}(s_i, s_f)$ from the initial state to the final state such that for all $\varrho' \in \Psi_{(S,TR,LB)}(s_i, s_f) : dist(\varrho) \leq dist(\varrho')$, the sequence of movements $\gamma = rev_{(N_1 \otimes N_2)}(\pi_2(\varrho))$ is an optimal alignment. ⌟

Proof. Theorem 4.3.5 shows that γ is an alignment. The following holds by definition: $dist(\varrho) = \sum_{tr \in \varrho} lc(rev_{(N_1 \otimes N_2)}(\pi_2(tr))) = \sum_{(x,y) \in \gamma} lc((x, y))$, i.e., the distance between two nodes is the same as the total likelihood cost of movements. Since ϱ is a shortest path from s_i to s_f, its distance yields the minimal likelihood cost of movements of all possible alignments. □

Theorem 4.4.2 provides a theoretical foundation to use shortest path algorithms in order to construct an optimal alignment, given a transition system of a product of an event net and a process net. As mentioned in Section 2.2, the $\mathbb{A}^\star$ algorithm is known to be the most efficient algorithm to compute such a shortest path in literature [50]. Given a directed graph, a source node, a target node, and a permissible function, the algorithm visits the least number of nodes among other shortest path algorithms to find a shortest path from the source node to the target node. However, the algorithm requires the number of paths with zero distance between nodes in the graph to be finite. Thus, given a transition system of a product of an event net and an easy sound process net, the algorithm only guarantees that a shortest path between two states in the transition system if it has a finite number of paths between states with total distance of zero. Checking such a constraint is computationally expensive as we need to construct the whole transition system and check possible paths for all pairs of states.

We take a pragmatic approach to solve this problem. The distance of an edge in a directed graph constructed from a transition system corresponds to the cost of movements represented by the label of the edge. If each edge in the graph has distance value above zero then the graph satisfies the requirement of the $\mathbb{A}^\star$ algorithm. Hence, *using a cost function of movements that assigns non-zero positive values to all movements is a sufficient condition to apply the $\mathbb{A}^\star$ algorithm*. Note that this is not a *necessary* condition. Given a trace, an easy sound process net, and a cost function for movements lc, we create another cost function lc' from lc that maps all movements to positive non-zero cost. For all movements, lc' returns the same cost as lc, added with a negligibly small cost $\epsilon \in I\!R^+$. For example, if we use the standard cost function, the new cost function assigns cost ϵ to all synchronous moves and moves on model (invisible transitions) instead of 0. The cost of moves on model (normal transitions) and moves on log is $\epsilon + 1$. The choice for the value of $\epsilon > 0$ depends on specific traces and process nets. As a guideline, one could use $\epsilon = \frac{highest\ cost\ of\ movements}{(lowest\ cost\ of\ movements)\cdot(size\ of\ trace)}$. However, ϵ should always be significantly smaller than all other costs. If the chosen value is too high then the obtained results may not be optimal with respect to the original cost function. If the chosen ϵ is too small, finding an optimal alignment may take longer. After getting an optimal alignment γ between the trace and the process net, $\epsilon \cdot |\gamma|$ can be substracted from the total cost of γ to get the "real" total cost of γ. Note that no new cost function needs to be constructed if the original cost function maps all movements to positive non zero values.

In practice, the performance of the $\mathbb{A}^\star$ algorithm also depends on the function it uses to estimate the shortest distance between all nodes in the directed graph to the closest target nodes. In Section 4.4.2, we provide a heuristic function that provide an estimation on such distance based on Petri net reachability theory.

4.4.2 Heuristic Function to Explore State Spaces Efficiently

Given a state and a target state in a transition system of the product of an event net for a trace and a process net, a function that always returns 0 is a permissible function for the labeled transition system of the product. However, we are interested in a precise estimation to guide the state space exploration of the $\mathbb{A}^\star$ algorithm. One of the possible ways is to provide a naive underestimation cost based on the minimum total cost required to reach a state that covers the final state of the event net, i.e., the minimum cost to perform moves on log/synchronous moves for all remaining activities in the trace.

Take for example the trace and the process net that are used to construct the transition system in Figure 4.4. Suppose that we want to have an underestimation of the shortest distance from state $[p_2, p'_2, p_4]$, i.e., the state after performing a synchronous move of activity *register* from the initial state, to the final state $[p_{12}, p'_6]$. The sequence of remaining activities in the trace that still need to be "replayed" is $\langle decide, register, send\ money, inform\ acceptance\rangle$. Intuitively, the activities must appear as either synchronous moves or moves on log in an optimal alignment between the trace and the net. Thus, the total minimum cost of movements of each activity yields a cost that underestimates the cost of the optimal alignment.

First of all, we show that each state in the transition system of a product net derived from an event net and a process net has exactly one place of the event net.

Lemma 4.4.3 (All markings of product net mark one place in the event net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be an event net over A and let N_2 be a process net over A. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$. For all $s \in S : |s_{\downarrow P_1}| = 1$. ⌟

Proof. It is trivial to see that for all markings $m \in RS(N_1, m_{i,1}) : |m| = 1$. According to Theorem 4.3.4, for all $s \in S : s_{\downarrow P_1} \in RS(N_1, m_{i,1})$. Hence, $|s_{\downarrow P_1}| = 1$. □

We formalize the naive permissible function described earlier in this section in Theorem 4.4.4. Note that since permissible functions are only used to estimate the distance to a single target node (i.e., the final state of a transition system), in the remainder of this thesis we omit the set of target nodes in all of the signatures of permissible functions.

Theorem 4.4.4 (Naive permissible function [8])
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be the event net of a trace $\sigma \in A^*$ over A, let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be an easy sound process net over A. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$. Let $lc : A^{\gg} \times T_2^{\gg} \to \mathbb{R}^+$ be a likelihood cost function of movements. Let $dist : TR \to \mathbb{R}^+$ be a distance function, such that for all $tr \in TR$, $dist(tr) = lc(rev_{(N_1 \otimes N_2)}(\pi_2(tr)))$.

A function $h : S \to \mathbb{R}$ maps the states of $TS(N_1 \otimes N_2)$ to real values such that for all $s \in S$:

$$h(s) = \min\left(\{\sum_{t_1 \in \varrho} \min\left(\{lc((\alpha_1(t_1), y)) \mid y \in T_2\}\right) \mid s_{\downarrow P_1} \xrightarrow{\varrho}_{N_1} m_{f,1}\}\right)$$

is a *permissible function* for (S, TR, LB) and $dist$. ⌟

Proof. We prove that the function is admissible by considering two possible cases:

- If there is no path from s to s_f, i.e., $\Psi_{(S,TR,LB)}(s, s_f) = \emptyset$ then any value provided by h is below $+\infty$.
- If a path $\varrho \in \Psi_{(S,TR,LB)}(s, s_f)$ exists, we consider two sub-cases: (1) If $s_{\downarrow P_1} = m_{f,1}$ then $h(s) = 0$. It is trivial to see that any function that returns 0 is admissible. (2) If $s_{\downarrow P_1} \neq m_{f,1}$, we know that N_1 is a sequence and $|s_{\downarrow P_1}| = 1$ (see Lemma 4.4.3). Thus, there exists a firing sequence $s_{\downarrow P_1} \xrightarrow{\varrho} m_{f,1}$, $\varrho \in T_1^*$. ϱ yields the remaining activities in σ that need to be replayed after state s. Therefore, $h(s)$ is the minimum total likelihood cost of movements to replay all of the activities. Hence, h is admissible.

The proof that h is consistent is as follows. For all edges $(s_1, tr, s_2) \in TR$, either $s_{1\downarrow P_1} = s_{2\downarrow P_1}$ or $s_{1\downarrow P_1} = s_{2\downarrow P_1}$. In the former case, $h(s_1) = h(s_2)$. In the latter case, tr yields either a move on log or a synchronous move (see Lemma 4.4.3). Since $h(s_1)$ is based on the minimum value of both of them, $h(s_1) \leq dist((s_1, tr, s_2)) + h(s_2)$ holds.

□

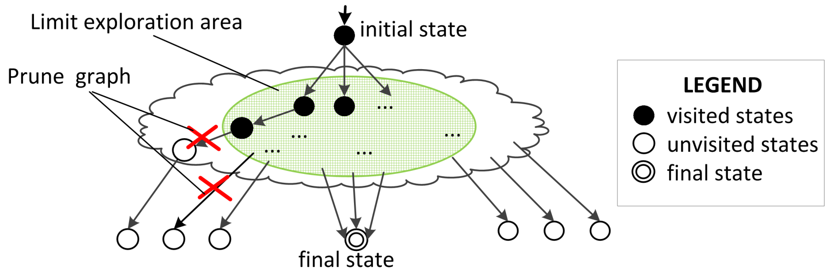


Figure 4.6: Two ways in which the marking equation is exploited: pruning exploration graph and limit exploration area.

Given a trace and a process net, to this point we propose a permissible function that only takes into account the number of "unreplayed activities" in the trace without exploiting information about the process net. Given a transition system, such a function may not be able to provide a precise estimation on the distance of each state in the transition system to the final state of the transition system during state space analysis. As a consequence, the $\mathbb{A}^\star$ algorithm may explore the states in a breadth-first-search manner and require a huge amount of memory as many states need to be visited and queued.

Therefore, we introduce a permissible function for the $\mathbb{A}^\star$ algorithm that provides a more precise underestimation of the remaining distance from each state in the system to its final state based on the marking equation of Petri nets [10]. The proposed function improves the efficiency of the $\mathbb{A}^\star$-based state space exploration in two ways (see Figure 4.6). First of all, the *state space is pruned* as for some states we can state with certainty that the final state is no longer reachable. If, according to the marking equation, the final state is no longer reachable, we do not need to explore successor states in the transition system. Second, by using the marking equation we can *provide a better underestimation of the total cost required to reach the final state*. This way, state space exploration can be limited to those states that most likely lead to the final state.

Pruning the Exploration Graph

To prune the exploration graph we exploit the well known *Petri net marking equation*. The following result can be found in any textbook on Petri nets, e.g., [119].

Theorem 4.4.5 (Reachability implies solution to marking equation)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let N be a Petri net over A, let $[\mathbf{N}]$ be the incidence matrix of N. Let $TS(N) = (S, TR, LB, s_i, s_f)$ be the transition system of N. For all $s, s' \in S, \varrho \in T^*$ such that $s \xrightarrow{\varrho} s'$, the following marking equation holds: $\vec{s} + [\mathbf{N}] \cdot \vec{\varrho} = \vec{s'}$ ⌟

Proof. See for example the proof in [119]. □

Theorem 4.4.5 states that reachability implies a solution. This implies that if no solution exists then state s' is not reachable from state s. Recall that we need to find a shortest path to the final node and all paths are firing sequences. Hence, we exploit this knowledge to identify nodes in the exploration graph from which the target node cannot be reached.

Theorem 4.4.6 (State pruning)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let N be a Petri net over A. Let $[\mathbf{N}]$ be the incidence

matrix of N. Let $TS(N) = (S, TR, LB, s_i, s_f)$ be the transition system of N. For all $s \in S$, if there is no solution $\vec{x}$ to $\vec{s} + [\mathbf{N}] \cdot \vec{x} = \vec{s_f}$, $\Psi_{(S,TR,LB)}(s, s_f) = \emptyset$. ⌟

Proof. We prove this theorem by contradiction. Suppose that there exists a state $s' \in S$ such that no solution to $\vec{s'} + [\mathbf{N}] \cdot \vec{x} = \vec{s_f}$, but there exists a path $\sigma \in \Psi_{(S,TR,LB)}(s', s_f)$ from state s' to final state s_f. Let $\varrho = \pi_2(\sigma)_{\downarrow T}$ be the firing sequence of σ. By definition, we know that $s \xrightarrow{\varrho} s_f$ and according to Theorem 4.4.5, $\vec{s} + [\mathbf{N}] \cdot \vec{\varrho} = \vec{s_f}$ holds. Thus, $\vec{x} = \vec{\varrho}$ is a solution.

□

The marking equation helps in pruning the exploration graph. However, we can also use the marking equation to provide a better underestimate of the remaining cost.

Limiting the State Space Exploration Area

Given a transition system, we showed that the existence of a solution to the marking equation for a state in the transition system (see Theorem 4.4.5) strongly correlates with the existence of a path from the state to the final state of the transition system. Next, we show that if such path exists, a solution to the marking equation with the minimum total cost yields a lower bound for the path distance. Since all transitions in the product of an event net and a process net represent movements, we define the cost of transitions in such a product based on the cost of movements.

Definition 4.4.7 (Cost of transitions)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let N_1 be an event net over A and let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be a process net over A. Let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2. Let $lc : A^{\gg} \times T_2^{\gg} \to \mathbb{R}^+$ be a likelihood cost function of movements. $c_{lc} : T_3 \to \mathbb{R}^+$ is the *cost of firing transitions*, defined by lc such that for all $t_3 \in T_3, c_{lc}(t_3) = lc(rev_{(N_1 \otimes N_2)}(t_3))$. ⌟

For simplicity, in the remainder of this section we use the cost of firing transitions directly instead of the likelihood cost of movements. Next, we show that marking equation provides lower bound for the total cost of movements.

Theorem 4.4.8 (Marking equation solution provides lower bound)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $[\mathbf{N}]$ be the incidence matrix of N. Let $TS(N) = (S, TR, LB, s_i, s_f)$ be the transition system of N. Let $c_{lc} : T \to \mathbb{R}^+$ be a cost function of firing transitions, and let $dist : TR \to \mathbb{R}^+$ be a distance function, such that for all $tr \in TR$, $dist(tr) = c_{lc}(\pi_2(tr))$.
For all states $s \in S$ where $\Psi_{(S,TR,LB)}(s, s_f) \neq \emptyset$

- let $v = \min\left(\{\sum_{1 \le j \le |T|} \vec{x}_j \cdot c_{lc}(T[j]) \mid \vec{s} + [\mathbf{N}] \cdot \vec{x} = \vec{s_f}\}\right)$ be the minimum value of total cost of solution to marking equation, and
- let $v' = \min(\{dist(\varrho) \mid \varrho \in \Psi_{(S,TR,LB)}(s, s_f)\})$ be the minimum distance of all paths from s to s_f.

The following statement holds: $v \le v'$ ⌟

Proof. We prove this theorem by first showing that a shortest path from s to s_f is also a solution to the marking equation. Suppose that $\sigma \in \Psi_{(S,TR,LB)}(s, s_f)$ is a shortest path from s to s_f. From Theorem 4.3.5, we know that $\varrho = \pi_2(\sigma)$ is a firing sequence from s to s_f, i.e., $s \xrightarrow{\varrho}_N s_f$. Hence, $\vec{s} + [\mathbf{N}] \cdot \vec{\varrho} = \vec{s_f}$ holds.

The total cost of solution to marking equation is the same as the distance of path (see the proof of Theorem 4.4.2). Therefore, $\sum_{1\leq j\leq|T|}\vec{\varrho}_j \cdot c_{lc}(T[j]) = \sum_{t\in\varrho} c_{lc}(t) = dist(\sigma)$. v is minimal, therefore $v \leq dist(\sigma)$ holds.

□

Given a state in a transition system, we know from Theorem 4.4.6 that if a solution to marking equation in Theorem 4.4.5 does not exist, there is no path to the final state. Hence, there is no point in exploring the successors of that state anymore. If a solution exists, we know a lower bound for the distance from the state to the final state (see Theorem 4.4.8). We use this knowledge to define a permissible underestimation function for the $\mathbb{A}^\star$ algorithm.

Theorem 4.4.9 (Precise permissible function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be the product of an event net and an easy sound process net over A. Let $[\![\mathbf{N}]\!]$ be the incidence matrix of N. Let $TS(N) = (S, TR, LB, s_i, s_f)$ be the transition system of N. Let $c_{lc} : T \to I\!R^+$ be a cost function of firing transitions. Let $dist : TR \to I\!R^+$ be a distance function, such that for all $tr \in TR : dist(tr) = c_{lc}(\pi_2(tr))$.

A function $h : S \to I\!R$ to the final state s_f such that for all $s \in S$, the following holds:

- $h(s) = +\infty$ if there is no solution to $\vec{s} + [\![\mathbf{N}]\!] \cdot \vec{x} = \vec{s_f}$, otherwise
- $h(s) = \min\left(\{\sum_{1\leq j\leq|T|} \vec{x}_j \cdot c_{lc}(T[j]) \mid \vec{s} + [\![\mathbf{N}]\!] \cdot \vec{x} = \vec{s_f}\}\right)$

is a permissible estimation function for (S, TR, LB) and $dist$.

┘

Proof. For all $s \in S$, if there is no solution to the marking equation then there is no path from s to s_f (see Theorem 4.4.6). Therefore, $h(s) = +\infty$ is an underestimation to the shortest distance from s to s_f. If there is a solution, we know that $h(s)$ is a lower bound for the distance from s to s_f (see Theorem 4.4.8). Thus, $h(s)$ is admissible.

To prove that h is also consistent, we use contradictions. Suppose that there exists an arc $(s_1, t, s_2) \in TR$ such that $h(s_1) > dist((s_1, t, s_2)) + h(s_2)$:

- If there is no solution to marking equation $\vec{s_2} + [\![\mathbf{N}]\!] \cdot \vec{x} = \vec{s_f}$ then $h(s_2) = +\infty$. Obviously, $h(s_1) \leq h(s_2)$ (contradiction).
- Suppose that $\vec{x}$ is a solution to marking equation $\vec{s_2} + [\![\mathbf{N}]\!] \cdot \vec{x} = \vec{s_f}$ that yields cost $h(s_2)$. Let $\vec{y}$ be a column vector of size $|T|$ where $\vec{y}_{idxOf(t,T)} = 1$ and $\vec{y}_i = 0$ for $1 \leq i \leq |T|, i \neq idxOf(t, T)$. We know that $s_1 \xrightarrow{t}_N s_2$. Thus, $\vec{s_2} = \vec{s_1} + [\![\mathbf{N}]\!] \cdot \vec{y}$. By substitution, $\vec{s_f} = \vec{s_2} + [\![\mathbf{N}]\!] \cdot \vec{x} = \vec{s_1} + [\![\mathbf{N}]\!] \cdot (\vec{x} + \vec{y})$.
  $h(s_1)$ is computed from a possible solution $\vec{z}$ to marking equation $\vec{s_1} + [\![\mathbf{N}]\!] \cdot \vec{z} = \vec{s_f}$ that yields a minimum value. Therefore, $h(s_1) \leq \sum_{1\leq j\leq|T|}(\vec{x}_j + \vec{y}_j) \cdot c_{lc}(T[j])$ and $h(s_1) \leq \sum_{1\leq j\leq|T|} \vec{x}_j \cdot c_{lc}(T[j]) + \sum_{1\leq k\leq|T|} \vec{y}_k \cdot c_{lc}(T[k])$. By definition, we know that $\sum_{1\leq j\leq|T|} \vec{y}_j \cdot c_{lc}(T[j]) = dist((s_1, t, s_2))$ and $\sum_{1\leq j\leq|T|} \vec{x}_j \cdot c_{lc}(T[j]) = h(s_2)$. Hence, $h(s_1) \leq dist((s_1, t, s_2)) + h(s_2)$ (contradiction).

□

Finding a solution for the marking equation with the minimum cost can viewed as an ILP problem [42, 157, 197]. Many approaches and software tools to solve ILP problems exist. Given a trace and a process net, we use the permissible function in Theorem 4.4.9 to guide state space exploration on transition systems constructed from the product of an event net and a process net, such as the one shown in Figure 4.4.

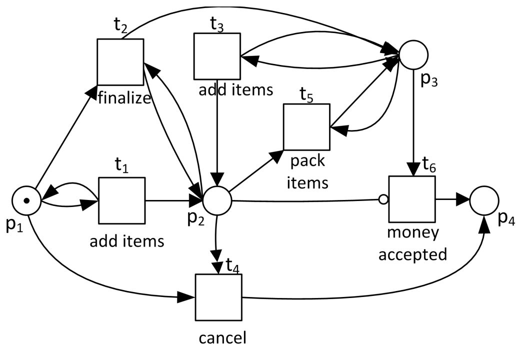


Figure 4.7: Example of a Petri net, extended with a reset arc (i.e., $r(t_4) = \{p_2\}$) and an inhibitor arc ($i(t_6) = \{p_2\}$). The same behavior cannot be expressed without such reset/inhibitor arcs.

4.4.3 Computing alignments of reset/inhibitor nets

Next to Petri nets, many other process modeling languages are used in practice. In this section, we extend the approach in Section 4.4.2 to deal with reset/inhibitor nets [200].

Figure 4.7 shows an example of a reset/inhibitor net. The net in the figure shows an online transaction process for an electronic bookstore. A customer can add as many items as possible to their cart (add items) before finalizing an order (finalize). After the order is finalized, ordered items are packed individually (pack items). All items of the order are sent to the customer immediately after they are packed and the payment for the order is accepted (money accepted). The customer may add more items after the order is finalized, but he can only cancel the order (cancel) before it is finalized. The process ends when all goods are sent (and the payment is accepted) or the process is cancelled.

Similar to Petri nets without reset/inhibitor arcs, the behavior of a reset/inhibitor net can be represented by a transition system. Since all theorems and lemmas that translate the problem of finding optimal alignments into shortest path problems are based on transition systems, i.e., Lemma 4.4.1 and Theorem 4.4.2, they still hold even for reset/inhibitor nets.

The general idea to construct an alignment between traces and reset/inhibitor nets is the same as the construction we described before for Petri nets. Given a trace and a reset/inhibitor net, we construct the product of the event net of the trace and the reset/inhibitor net in a similar way as constructing the product of two Petri nets. The only difference is that we also keep the reset/inhibitor arcs in the event net of the trace and add new reset/inhibitor arcs to synchronous transitions whose process net transitions are connected to reset/inhibitor arcs.

Figure 4.8 shows the product of the event net of a trace $\sigma_2 = \langle \textit{add items}, \textit{cancel}, \textit{add items}, \textit{finalize}, \textit{pack items}, \textit{pack items}, \textit{money accepted} \rangle$ and the reset/inhibitor net in Figure 4.7. All reset and inhibitor arcs in the original reset/inhibitor net are preserved. Synchronous transitions that are constructed from transitions that are connected to reset/inhibitor arcs also have reset/inhibitor arcs (e.g., synchronous transitions (t'_2, t_4) and (t'_7, t_6)).

Definition 4.4.10 (Product of two reset/inhibitor nets)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1}, r_1, i_1)$ and $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2}, r_2, i_2)$ be two reset/inhibitor nets over A. The *product* of N_1 and

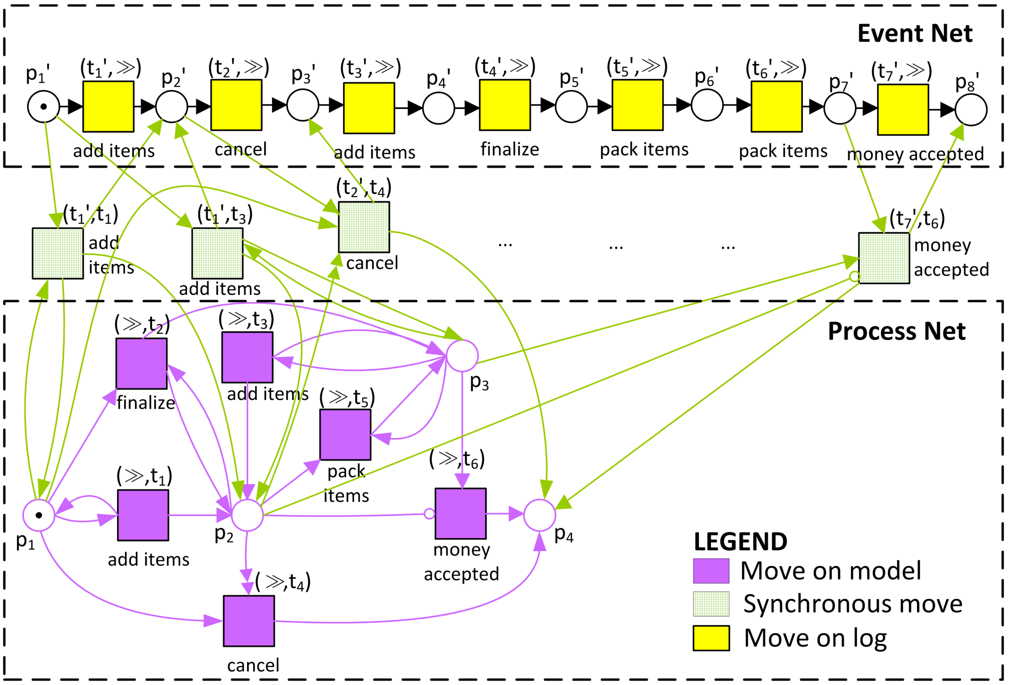


Figure 4.8: Product of event net of trace $\sigma_2 = \langle$*add items*, *cancel*, *add items*, *finalize*, *pack items*, *pack items*, *money accepted*$\rangle$ and the reset/inhibitor net in Figure 4.7.

N_2 is a reset/inhibitor net $N_3 = N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3}, r_3, i_3)$, such that

- $(P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3}) = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1}) \otimes (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$, i.e., places, arcs, and transitions are created using the product of two ordinary Petri nets (see Definition 4.3.2),
- $r_3 : T_3 \to \mathcal{P}(P_3)$ and $i_3 : T_3 \to \mathcal{P}(P_3)$ are the reset function and the inhibitor function respectively, such that
  - for all $(t_1, \gg) \in T_3$ where $t_1 \neq \gg, r_3((t_1, \gg)) = r_1(t_1)$ and $i_3((t_1, \gg)) = i_1(t_1)$,
  - for all $(\gg, t_2) \in T_3$ where $t_2 \neq \gg, r_3((\gg, t_2)) = r_2(t_2)$ and $i_3((\gg, t_2)) = i_2(t_2)$,
  - for all $(t_1, t_2) \in T_3$ where $t_1 \neq \gg$ and $t_2 \neq \gg, r_3((t_1, t_2)) = r_1(t_1) \cup r_2(t_2)$ and $i_3((t_1, t_2)) = i_1(t_1) \cup i_2(t_2)$

┘

The approach to prune the state space and direct state space exploration using the marking equation of Petri nets cannot be applied directly, because the equation is based on the incidence matrix that by definition ignores reset/inhibitor arcs. However, given a product of an event net and a reset/inhibitor net, we only need values that *underestimate* the actual distance from states of its transition system to its final state. Note that such a product is also a reset/inhibitor net. Therefore, we propose a general idea to estimate the actual distance of states of reset/inhibitor net using a Petri net with some cost function and constraints that can be related to the original reset/inhibitor net and its cost to fire transitions. We call such a net an *estimation net*.

Definition 4.4.11 (Estimation net and cost)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P, T_1, F_1, \alpha_1, m_i, m_f, r, i)$ be a reset/inhibitor net over A. Let $c_{lc_1} : T_1 \to I\!R^+$ be the cost function of firing transitions in T_1. Let $N_2 = (P, T_2, F_2, \alpha_2, m_i, m_f)$ be a Petri net over A and let $c_{lc_2} : T_2 \to I\!R$ be a cost function of firing transitions in T_2.

N_2 with cost function c_{lc_2} is an *estimation of* N_1 with cost function c_{lc_1} if for all markings $m, m' \in RS(N_1, m_i)$ and any firing sequence $\varrho_1 \in T_1^*$ such that $m \xrightarrow{\varrho_1}_{N_1} m'$, there exists a sequence $\varrho_2 \in T_2^*$ where:

- $m \xrightarrow{\varrho_2}_{N_2} m'$, i.e., the same marking is reachable in N_2, and
- $\sum_{t_2 \in \varrho_2} c_{lc_2}(t_2) \leq \sum_{t_1 \in \varrho_1} c_{lc_1}(t_1)$, i.e., firing ϱ_2 yields less or equal cost to firing ϱ_1.

┘

Given a reset/inhibitor net and the costs of firing its transitions, an estimation net of the original reset/inhibitor net and the cost function are used to provide an underestimation to the minimum distance between states of the reset/inhibitor net.

Theorem 4.4.12 (Estimation net and cost provide permissible underestimation)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P, T_1, F_1, \alpha_1, m_i, m_f, r, i)$ be an easy sound reset/inhibitor net over A, let $TS(N_1) = (S, TR, LB, s_i, s_f)$ be the transition system of N_1, let $c_{lc_1} : T_1 \to I\!R^+$ be the cost function of firing transitions in T_1. Let $dist_1 : TR \to I\!R^+$ be the distance of edges such that for all $tr \in TR$, $dist_1(tr) = c_{lc_1}(\pi_2(tr))$.

Let $N_2 = (P, T_2, F_2, m_i, m_f)$ be an estimation net of N_1 with cost function $c_{lc_2} : T_2 \to I\!R$.

A function $h : S \to I\!R$ such that for all $s \in S$:

- $h(s) = +\infty$ if there is no firing sequence $\varrho \in T_2^*$ such that $s \xrightarrow{\varrho}_{N_2} s_f$, and
- $h(s) = \min\left(\{\sum_{t_2 \in \varrho} c_{lc_2}(t_2) \mid s \xrightarrow{\varrho}_{N_2} s_f \land \varrho \in T_2^*\}\right)$ otherwise.

is a permissible function for (S, TR, LB) and $dist_1$ to estimate the distance from all states S to s_f. ┘

Proof. First, we show that the function is admissible. We consider two possible cases:

- Suppose that $\Psi_{(S,TR,LB)}(s, s_f) \neq \emptyset$. By definition, for all firing sequences $\varrho_1 \in T_1^*$ such that $s \xrightarrow{\varrho_1}_{N_1} s_f$, there exists $\varrho_2 \in T_2^*$ such that $s \xrightarrow{\varrho_2}_{N_2} s_f$. Furthermore, $\sum_{t_2 \in \varrho_2} c_{lc_2}(t_2) \leq \sum_{t_1 \in \varrho_1} c_{lc_1}(t_1)$. This implies $h(s) \leq \sum_{t_2 \in \varrho_2} c_{lc_2}(t_2)$, hence $h(s) \leq \sum_{t_1 \in \varrho_1} c_{lc_1}(t_1)$. Since any firing sequence in N_1 from s to s_f yields a path from s to s_f in (S, TR, LB), $h(s)$ underestimate the distance from s to s_f.
- Suppose that $\Psi_{(S,TR,LB)}(s, s_f) = \emptyset$, i.e., state s_f is not reachable from s in N_1. Any value of $h(s) \leq +\infty$. Thus, h is admissible.

Second, we show that h is also consistent using contradictions. Suppose that there exists an edge $(s_1, t_1, s_2) \in TR$ where $h(s_1) > dist((s_1, t_1, s_2)) + h(s_2)$. Let $\varrho \in T_2^*$ be a firing sequence such that $s_2 \xrightarrow{\varrho}_{N_2} s_f$ and ϱ yields the minimum cost of firing sequence from s_2 to s_f. We know that there exists a firing sequence $\varrho' \in T_2^*$ such that $s_1 \xrightarrow{\varrho'}_{N_2} s_2$ and $\sum_{t_2 \in \varrho'} c_{lc_2}(t_2) \leq dist((s_1, t_1, s_2))$. Furthermore, $s_1 \xrightarrow{\varrho' \cdot \varrho} s_f$. By definition, $h(s_1) \leq \sum_{t' \in \varrho'} c_{lc_2}(t') + \sum_{t \in \varrho} c_{lc_2}(t)$. Hence, $h(s_1) \leq dist((s_1, t_1, s_2)) + \sum_{t \in \varrho} c_{lc_2}(t)$ (contradiction).

□

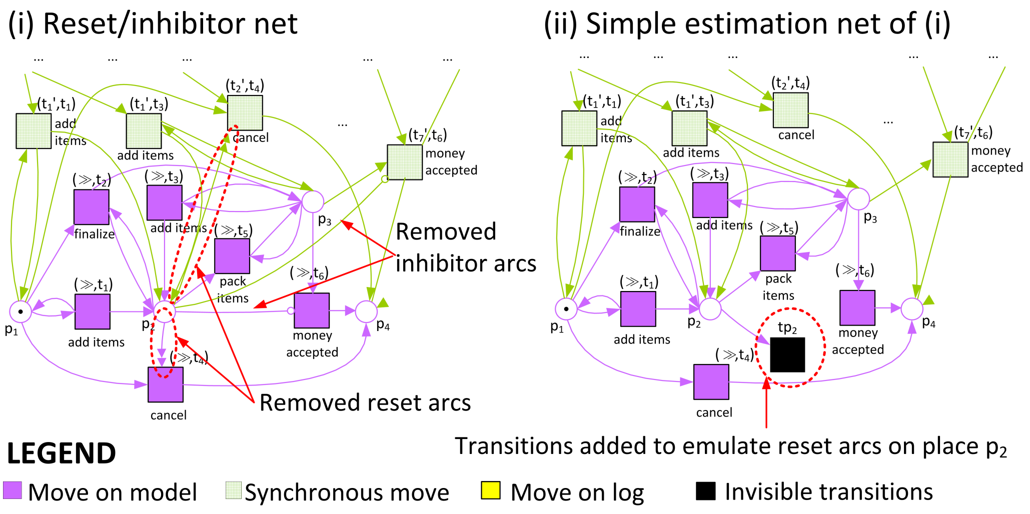


Figure 4.9: An estimation net of the reset/inhibitor net shown in Figure 4.8: Inhibitor arcs $i((\gg, t_6))$ and $i((t_7', t_6))$ are removed and both reset arcs $r((\gg, t_4))$ and $((t_2', t_4))$ are replaced by a sink invisible transition (t_{p2}).

Given an event net, a process net with reset/inhibitor arcs, and a cost of movements, there are possibly many estimation nets of the product between the event net and the process net that one can construct. We encode reset arcs explicitly as invisible transitions that take tokens from places connected to reset arcs, i.e., *reset places*. Figure 4.9 shows a reset/inhibitor net and its simple estimation net. Transitions and places of the original net are preserved and an extra transition is added for each reset place in the original net. Furthermore, inhibitor arcs are removed. Intuitively, all behavior allowed by the original net is allowed by the estimation net, but not necessarily the other way around. Furthermore, we use a cost function where the cost of firing transitions is the same as the cost of firing the original transitions, while the costs of firing the extra transitions are 0. Note that a zero cost function is allowed because the net is only used to provide an estimation. We show that such a net and cost is an estimation.

First, we define such an estimation net and cost function as follows:

Definition 4.4.13 (Simple estimation Petri nets and cost)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f, r, i)$ be a process net over A, and let $c_{lc} : T \to \mathbb{R}^+$ be a cost function of firing transitions. $N' = (P, T \cup T', F', \alpha', m_i, m_f)$ is the *simple estimation net* of N where

- $T' = \{t_p \mid p \in \bigcup_{t' \in T} r(t')\}$, i.e., an extra transition is added for each reset place in N,
- $F' : (P \times (T \cup T')) \cup ((T \cup T') \times P) \to \mathbb{N}$ is a flow relation returning the weight of arcs, such that
  - for all $e \in (P \times T) \cup (T \times P), F'(e) = F(e)$,
  - for all $t_p \in T'$, the following holds:
    * $F'(t_p, p) = 0$,
    * $F'(p, t_p) = 1$, and
    * For all $p' \in P \setminus \{p\}, F(t_p, p') = F(p', t_p) = 0$

- $\alpha' : (T \cup T') \to A^\tau$ is a labeling function such that for all $t \in T : \alpha'(t) = \alpha(t)$ and for all $t' \in T' : \alpha'(t) = \tau$

Furthermore, $c'_{lc} : (T \cup T') \to I\!R$ is a estimation cost function for N' such that for all $t \in T : c'_{lc}(t) = c_{lc}(t)$ and for all $t' \in T' : c'_{lc}(t') = 0$.

⌟

We show that the simple estimation net and cost satisfy the requirements stated in Definition 4.4.3.

Theorem 4.4.14 (Simple estimation net and cost satisfy requirements)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f, r, i)$ be an easy sound reset/inhibitor net over A, and let $c_{lc} : T \to I\!R^+$ be the costs of firing transitions in N. Let $N' = (P, T \cup T', F', \alpha', m_i, m_f)$ be the simple estimation net of N and let $c'_{lc} : (T \cup T') \to I\!R$ be its cost function as defined in Definition 4.4.13.
N' with cost function c'_{lc} is an estimation of N with cost function c_{lc}. ⌟

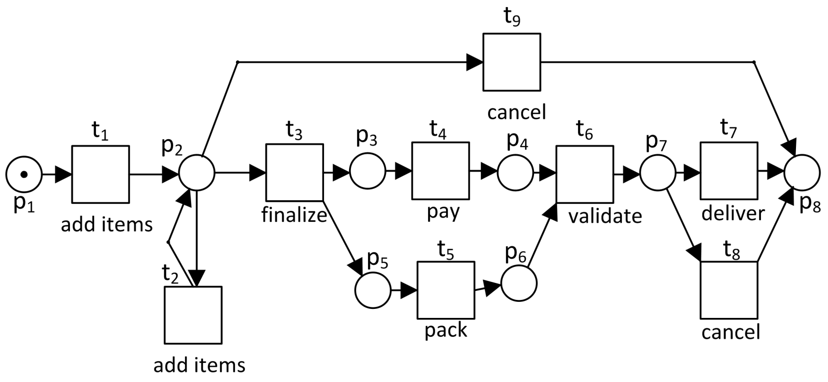
Proof. For all transitions $t \in T$ enabled at the initial marking (i.e., $(N, m_i)[t\rangle$), if t is not connected to any reset arc then the firing of t in N, i.e., $m_i \xrightarrow{t}_N m$, can be mimicked by firing the same transition in N' from the same marking m_i, i.e., $m_i \xrightarrow{t}_{N'} m$. If t is connected to some reset arcs in N, the firing t can be mimicked in N' by firing the same transition, i.e., $m_i \xrightarrow{t} m''$, and firing some extra invisible transitions in N' to remove places connected to the reset arcs until marking m' is reached. Iteratively, we can show that for all reachable markings $m_x \in RS(N, m_i)$ there exists a firing sequence in N' to reach the same markings. Since the cost of firing all transitions in T is the same for both c_{lc} and c'_{lc} and the cost of firing any of the extra transition $t' \in T'$ is 0, it is trivial to see that any firing sequence in N can be mimicked in N' with exactly the same total cost.

□

Theorem 4.4.14 shows that simple estimation nets and cost functions yield permissible heuristic functions for reset/inhibitor nets. We perform state space analysis as we did before, but instead of using the marking equation of the reset/inhibitor net to provide estimation, we use the marking equation of its estimation net. Given a trace and a reset/inhibitor net, for all visited states s in the net, the estimation of the remaining distance from s to its final state is the same as the estimation of remaining distance from the same state s in its estimation net to its final state. Hence, the $\mathbb{A}^\star$ algorithm and the marking equation can both be used to compute optimal alignments.

4.5 Computing an Optimal Prefix Alignment

By definition, optimal alignments penalize traces that do not reach proper termination as described by process models. However, if traces are known in advance to be not completed yet (i.e., new activities may be appended to the traces), optimal alignments may mistakenly consider the incompleteness of such traces as deviations. Take for example a trace $\sigma_3 = \langle \mathit{add\ items}, \mathit{finalize} \rangle$ of the process net in Figure 4.10. Suppose that the trace is not completed yet, i.e., more activities may occur for the same trace. Using the standard cost function, an optimal alignment between the trace and the model is shown in Figure 4.10. Notice that the occurrence of activity finalize in the trace is considered as a deviation (move on log), while it should not be the case as new events may occur to make the trace perfectly fitting (e.g., the sequence of activities $\langle \mathit{pay}, \mathit{pack}, \mathit{validate}, \mathit{deliver} \rangle$).



$\gamma =$	*add items*	*finalize*	$\gg$
	add items		*cancel*
	t_1	$\gg$	t_9

Figure 4.10: Top: Process net of an online transaction in an electronic bookstore, and **Bottom:** an optimal alignment between an incompleted trace $\sigma_3 = \langle add\ items, finalize \rangle$ and the process net, showing that the occurrence of finalize in the trace is a deviation.

Therefore, we define a less strict notion of alignments, called *prefix alignments*. The main idea of prefix alignments is that incomplete traces are not penalized for not having reached the final marking. Thus, given a trace and a process model, the absence of activities in the trace that must occur according to process models to terminate properly is ignored. Prefix alignments are defined as follows:

Definition 4.5.1 ((Optimal) Prefix Alignment)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. A *prefix alignment* $\gamma \in (A^{\gg} \times T^{\gg})^*$ between σ and N is a movement sequence such that:

- $\pi_1(\gamma)_{\downarrow A} = \sigma$, i.e. its sequence of movements in the trace (ignoring $\gg$) yields the trace, and
- There exists a complete firing sequence $m_i \xrightarrow{\varrho} m_f$ such that $\pi_2(\gamma)_{\downarrow T} < \varrho$, i.e. its sequence of movements in the model (ignoring $\gg$) yields a *prefix* of a complete firing sequence of N.

$\Lambda_{\sigma,N}$ is the set of all prefix alignments between a trace σ and a Petri net N.

Let $lc : A^{\gg} \times T^{\gg} \to \mathbb{R}$ be a likelihood cost function for movements. A prefix alignment $\gamma \in \Lambda_{\sigma,N}$ is *optimal* if and only if for all $\gamma' \in \Lambda_{\sigma,N}, \sum_{(a,b)\in\gamma} lc((a,b)) \leq \sum_{(a',b')\in\gamma'} lc((a',b'))$. ⌟

Figure 4.11 shows an optimal prefix alignment between trace $\sigma_3 = \langle add\ items, finalize \rangle$ and the process in Figure 4.10 according to the standard cost function. Compare the optimal prefix alignment with the optimal alignment shown in Figure 4.10. In the prefix alignment, the occurrence of activity finalize is not considered as a deviation. Instead, it is considered as a synchronous move because the trace is a prefix of a sequence of activities, yielded by a complete firing sequence of the net, i.e., trace $\langle add\ items, finalize \rangle$ is a prefix of $\langle add\ items, finalize, pay, pack, validate, deliver \rangle$, resulting from the complete firing sequence $\langle t_1, t_3, t_4, t_5, t_6, t_7 \rangle$ of the process net.

$$\gamma' = \begin{array}{|c|c|} \textit{add items} & \textit{finalize} \\ \hline \textit{add items} & \textit{finalize} \\ t_1 & t_3 \end{array}$$

Figure 4.11: An optimal prefix alignment between the uncompleted trace $\sigma_2 = \langle \textit{add items}, \textit{finalize} \rangle$ and the net in Figure 4.10, showing that there is no deviation in the trace.

Given a trace and a process net, in Section 4.3 we showed how to explicitly model the set of all movements that can be performed in order to construct an optimal alignment between the trace and the process net. We can use a similar approach as the one proposed in Section 4.3 to construct an optimal prefix alignment between a trace and a process net where the final marking can be reached from any reachable marking.

Definition 4.5.2 (Petri Net with option to complete)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. N has *option to complete* if and only if for all $m \in RS(N, m_i) : m_f \in RS(N, m)$. ⌟

It is trivial to see that a Petri net with option to complete is also an easy sound Petri net. We show that a firing sequence of the product of an event net and a process net defines a prefix alignment.

Theorem 4.5.3 (Firing sequences define prefix alignments)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A and let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be its event net. Let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be a Petri net over A with option to complete, and let $N_1 \otimes N_2 = (P_3, T_3, F_3, \alpha_3, m_{i,3}, m_{f,3})$ be the product of N_1 and N_2. For all firing sequences $m_{i,3} \xrightarrow{\varrho}_{(N_1 \otimes N_2)} m, \varrho \in T_3^*$, if $m_{\downarrow P_1} = m_{f,1}$ then $rev_{(N_1 \otimes N_2)}(\varrho)$ is a prefix alignment. ⌟

Proof. Theorem 4.3.5 proves that $rev_{(N_1 \otimes N_2)}(\varrho)$ is a movement sequence. An event net consists of a sequence of transitions and therefore the only way to reach the final marking $m_{f,1}$ from $m_{i,1}$ is by firing all transitions in the net (see Definition 4.3.1). In the product net, a marking m such that $m_{\downarrow P_1} = m_{f,1}$ can only be reached if transitions derived from the event net are fired. This implies $\pi_1(rev_{(N_1 \otimes N_2)}(\varrho))_{\downarrow A} = \sigma$. N_2 has option to complete, therefore there exists $\varrho' \in T_2^*$ such that $m_{\downarrow P_2} \xrightarrow{\varrho'}_{N_2} m_{f,2}$. Therefore, there exists a firing sequence $\varrho'' \in T_3^*$ such that $m \xrightarrow{\varrho''}_{N_3} m_{f,3}$ (see Theorem 4.3.4). We know that $m_{i,3} \xrightarrow{\varrho \cdot \varrho''} m_{f,3}$ and $rev_{(N_1 \otimes N_2)}(\varrho \cdot \varrho'')$ is an alignment (see Theorem 4.3.5). Hence, $rev_{(N_1 \otimes N_2)}(\varrho)$ is a prefix alignment. □

Take for example the trace $\sigma_1 = \langle \textit{register}, \textit{decide}, \textit{register}, \textit{send money}, \textit{inform acceptance} \rangle$ and the process net shown in Figure 4.1. The process net has option to complete. Figure 4.12 shows an excerpt of the product between the event net of the trace and the process net. A prefix alignment can be obtained from any firing sequence ϱ of the product net that yields a marking m, such that the projection of m to places of the event net yields its final marking (i.e., $[p'_6]$).

To compute such a firing sequence, we use a similar state space exploration approach based on the $\mathbb{A}^\star$ algorithm as the one proposed in Section 4.4. Given a transition system of a product of an event net and a process net, the target nodes that yield prefix alignments are all markings of the product reachable from its initial marking that cover the final marking of the event net (i.e., $[p'_6]$ in Figure 4.12). In the running example, the set of target nodes is $\{[p_1, p'_6], [p_2, p_4, p'_6], [p_3, p_4, p'_6], \ldots\}$. Figure 4.13 shows the transition

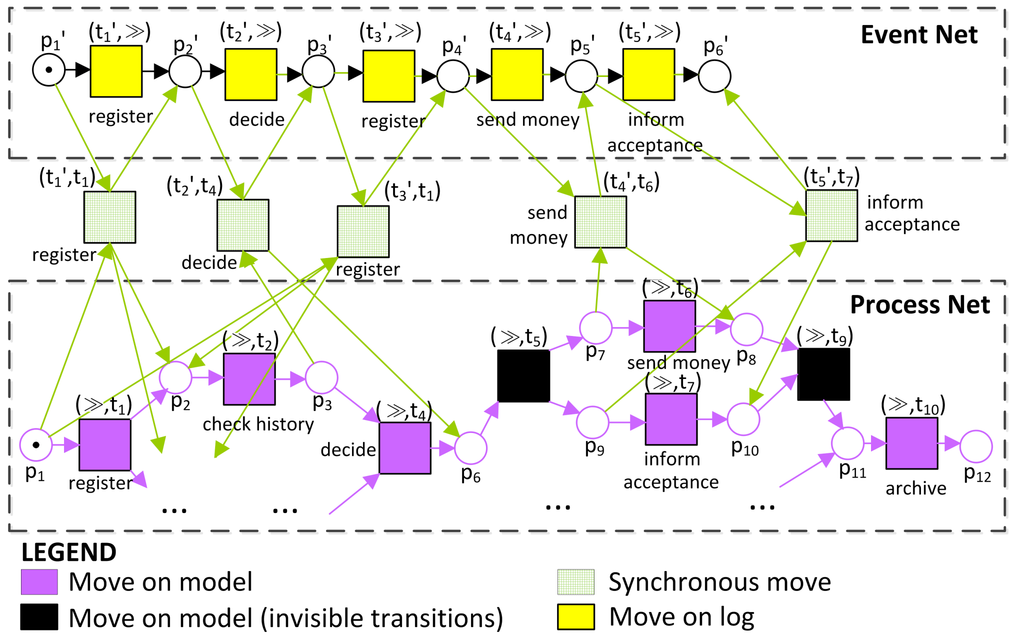


Figure 4.12: Excerpt of the product of the event net of trace $\sigma_1 = \langle \mathit{register}, \mathit{decide}, \mathit{register}, \mathit{send\ money}, \mathit{inform\ acceptance} \rangle$ and the process net in Figure 4.3. A firing sequence from the initial marking that leads to a marking m that covers $[p'_6]$ yields a prefix alignment.

system of the product net in Figure 4.12. Colored states are the set of all states that cover the final marking of the event net $[p'_6]$. Note that for each colored state, there is a path from the state to the final state of the transition system.

Similar to the steps followed in Section 4.4, first we prove that a path from the initial state of the product net to the target nodes exists.

Lemma 4.5.4 (A path to reach the final marking of event nets exists)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ and $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be an event net and a process net over A with option to complete respectively. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$. Let $S_f = \{s \in S \mid s_{\downarrow P_1} = m_{f,1}\}$ be the set of all states that yields the final marking of N_1.

For all $s \in S_f : \Psi_{(S,TR,LB)}(s_i, s) \neq \emptyset$, i.e., there exists a path from the initial state to each state that yields the final marking of N_1. ⌟

Proof. For all $s \in S_f, s \in RS(N_1 \otimes N_2, s_i)$ and therefore a firing sequence $s_i \xrightarrow{\varrho}_{(N_1 \otimes N_2)} s$ exists. By definition of transition system (see Definition 2.5.2), there exists a path $\langle (s_i, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_n, \varrho[|\varrho|], s) \rangle \in \Psi_{(S,TR,LB)}(s_i, s)$. Therefore, such a path exists. □

For all process nets with option to complete, we show that the $\mathbb{A}^\star$ provides a firing sequence that yields an optimal prefix alignment.

Theorem 4.5.5 (Shortest paths yield prefix optimal alignments)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N_1 = (P_1, T_1, F_1, \alpha_1, m_{i,1}, m_{f,1})$ be the event net of a trace $\sigma \in A^*$ over A. Let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be a process net over A with option to complete. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition

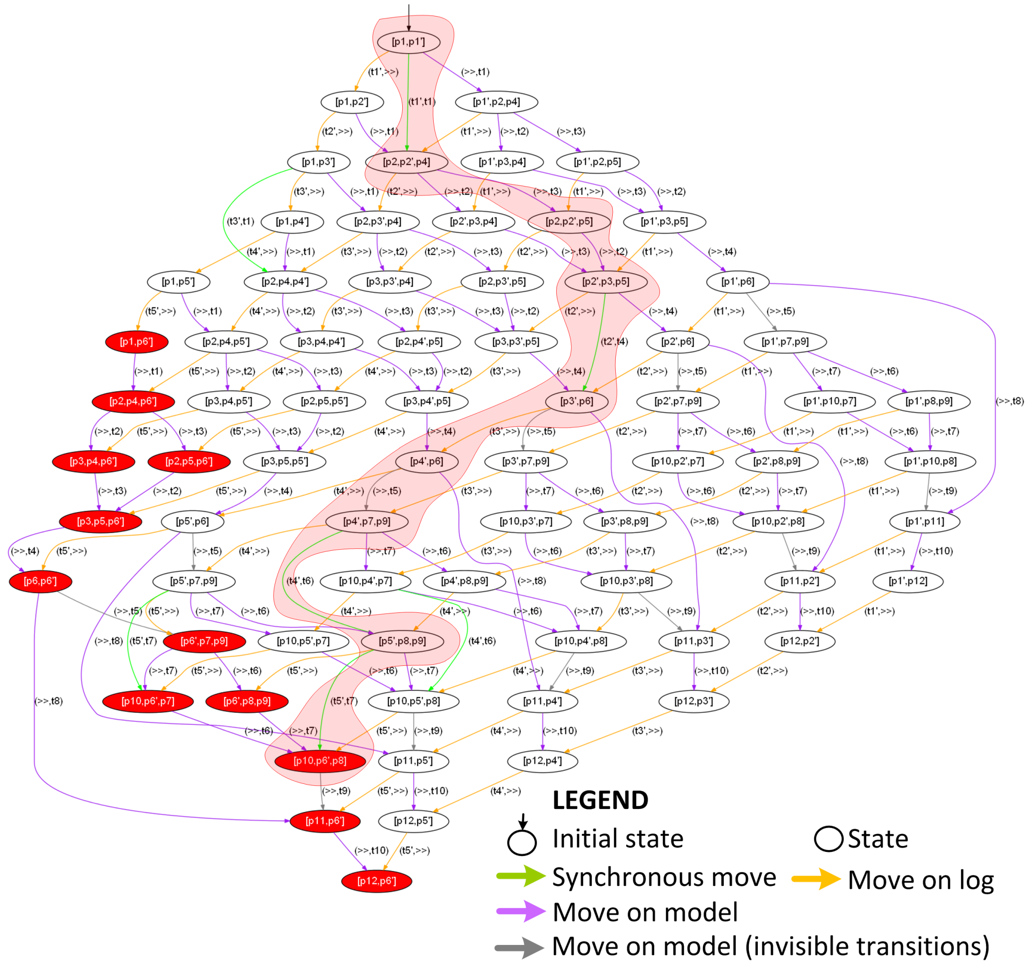


Figure 4.13: The transition system of the product between the event net of trace $\sigma_1 = \langle \mathit{register}, \mathit{decide}, \mathit{register}, \mathit{send\ money}, \mathit{inform\ acceptance} \rangle$ and the model in Figure 4.1. All colored states are markings that cover the final marking of the event net, i.e., marking $[p'_6]$. The highlighted path from the initial state to state $[p'_6, p_8, p_{10}]$ yields the optimal prefix alignment shown in Figure 4.14.

system of $N_1 \otimes N_2$. Let $lc : A^{\gg} \times T^{\gg} \rightarrow I\!R^+$ be a likelihood cost function of movements. Let $dist : TR \rightarrow I\!R^+$ be a distance function, such that for all $tr \in TR$, $dist(tr) = lc(rev_{(N_1 \otimes N_2)}(\pi_2(tr)))$.

Let $S_f = \{s \in S \mid s_{\downarrow P_1} = m_{f,1}\}$ be the set of states (marking) of TS that yields the final marking $m_{f,1}$ of N_1 and let $\varrho \in TR^*$ be a shortest path from s_i to any state in S_f, i.e., for all state $s_f \in S_f, \varrho' \in \Psi_{(S,TR,LB)}(s_i, s_f) : \sum_{tr \in \varrho} dist(tr) \leq \sum_{tr' \in \varrho'} dist(tr')$. The sequence of movements $rev_{(N_1 \otimes N_2)}(\pi_2(tr))$ is an optimal prefix alignment. ⌟

Proof. For all $s_f \in S_f$, by definition $s_{f\downarrow P_1} = m_{f,1}$ and there is no transition $t \in T_3 \cap (T_1 \times T_2^{\gg})$ can be fired. Furthermore, $m_{f,2} \in RS(N_2, s_{f\downarrow P_2})$ implies $m_{f,3} \in RS(N_3, s_f)$ (see Theorem 4.3.4). Thus, according to Theorem 4.3.5 $rev_{(N_1 \otimes N_2)}(\pi_2(\varrho))$ is a prefix alignment. Furthermore, since the distance of a path is the same as the total likelihood cost of an alignment constructed from the path (see Theorem 4.4.2), $rev_{(N_1 \otimes N_2)}(\pi_2(\varrho))$ is an optimal prefix alignment. □

register	$\gg$	$\gg$	*decide*	*register*	$\gg$	*send money*	*inform acceptance*
register	*check history*	*check cause*	*decide*			*send money*	*inform acceptance*
t_1	t_2	t_3	t_4	$\gg$	t_5	t_6	t_7

Figure 4.14: An optimal prefix alignment between trace $\sigma_1 = \langle register, decide, register, send\ money, inform\ acceptance\rangle$ and the model in Figure 4.1.

The colored path in Figure 4.13 shows a shortest path from the initial state to any target node of the transition system. The movement sequence constructed from the path is shown in Figure 4.14. The total likelihood cost of the optimal prefix alignment in Figure 4.14 is 3.

To this point, we have not discussed a permissible heuristic function that maps each state of a transition system to a value that underestimates the shortest distance from the state to reach any of the predefined target nodes. It is easy to see that a function that always returns zero for all states in the transition system provides such an underestimation. The naive cost function mentioned in Theorem 4.4.4 also provides such an underestimation.

Theorem 4.5.6 (Naive permissible underestimation function for prefix alignment) Let $A \subseteq \mathcal{A}$ be a set of activities. Let N_1 be the event net of a trace $\sigma \in A^*$ over A. Let $N_2 = (P_2, T_2, F_2, \alpha_2, m_{i,2}, m_{f,2})$ be a process net over A with option to complete. Let $TS(N_1 \otimes N_2) = (S, TR, LB, s_i, s_f)$ be the transition system of $N_1 \otimes N_2$. Let $lc : A^{\gg} \times T^{\gg} \to \mathbb{R}^+$ be a likelihood cost function of movements. Let $dist : TR \to \mathbb{R}^+$ be a distance function, such that for all $tr \in TR$, $dist(tr) = lc(rev_{(N_1 \otimes N_2)}(\pi_2(tr)))$. Let $S_f = \{s \in S \mid s_{\downarrow P_1} = m_{f,1}\}$ be the set of states (marking) of TS that yield the final marking of N_1.

A function $h : S \times \mathcal{P}(S) \to \mathbb{R}$ maps the states of $TS(N_1 \otimes N_2)$ to real values such that for all $s \in S$:

$$h(s, S_f) = \sum_{idxOf(s_{\downarrow P_1}[1], P_1) \leq i < |P_1|} \min\left(\{lc((\sigma[i], x)) \mid x \in T^{\gg}\}\right)$$

is a *permissible function* for (S, TR, LB) and $dist$ that provides an underestimation of the shortest distances to S_f. ⌟

Proof. The proof of this theorem is the same as the proof of Theorem 4.4.4. Intuitively, given a trace, a Petri net, and a likelihood cost function, each activity in the trace needs to be "replayed" as either a move on log or a synchronous move. For each state in the transition system, the total minimum likelihood cost of only considering the remaining activities to be "replayed" (without considering move on models) provides an underestimation for the "real" minimum cost (i.e., the function is admissible). Furthermore, using the same approach as the proof in Theorem 4.4.4, we know that this function is consistent.

□

The approach to compute optimal prefix alignments explained in this section is *limited to process nets with option to complete*. In [6], we propose an approach to compute optimal prefix alignments for easy sound process nets based on Petri net coverability theory. Given a trace and a process net, if the upper bound of total cost of deviations

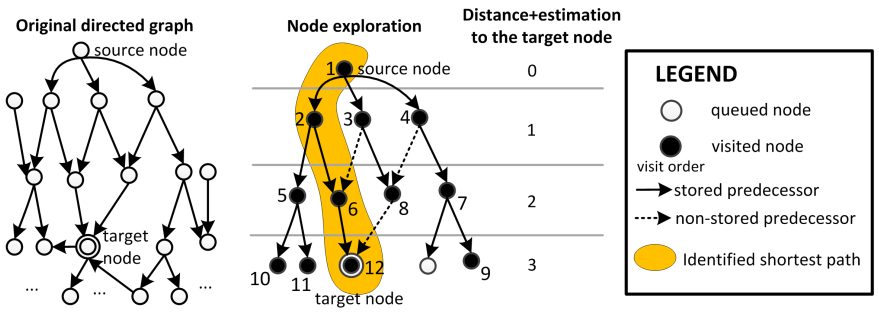


Figure 4.15: Left: a directed graph where all arcs have distance 1, and **Right:** The exploration graph obtained in the last iteration of the $\mathbb{A}^\star$ algorithm that yields a shortest path from the source node to the target node of the graph using a permissible underestimation function that always returns 0 for all nodes (as shown previously in Figure 2.1).

between the trace and the net is known in advance, a binary search in combination with marking coverability analysis of product of two Petri nets can be performed to compute an optimal prefix alignment between the trace and the process net. However, the computational complexity of this approach is much higher than the one presented in this section. The computation complexity of marking coverability in a Petri net is exponential to the size of the net. On top of that, the binary search complexity is logarithmic to the value of the upper bound. The complexity of the approach proposed in this section is the same as the complexity of the $\mathbb{A}^\star$ algorithm, which is exponential only to the size of the shortest path. Thus, the approach proposed in this section outperforms our earlier approach in [6].

4.6 Computing All Optimal (Prefix) Alignments and Representatives

Section 4.4 and Section 4.5 show how the $\mathbb{A}^\star$ algorithm can be used to compute optimal alignments and optimal prefix alignments. Given a trace and a process net, an optimal (prefix) alignment provides some diagnostics on the type of deviations that occur between them. However, there can be more than one optimal (prefix) alignment between them. To get a comprehensive diagnostics on all possible deviations, it may be necessary to compute all optimal (prefix) alignments.

We extend the original $\mathbb{A}^\star$ algorithm to compute the set of all shortest paths in a directed graph. Given a directed graph, a source node, and a set of target nodes, the $\mathbb{A}^\star$ algorithm uses a priority queue to store candidate nodes to be explored iteratively. In each iteration, the best candidate node in the priority queue is the one with the shortest distance from the source node after adding the underestimate for the remaining costs. This way, the algorithm never visits a less promising candidate node before all better candidates in the queue are visited. Suppose that the first best candidate node in the priority queue that is also a target node has a shortest distance value of x from the source node, we know that other shortest paths also have a distance value of x. We exploit this in order to compute all shortest paths from the source node to any of the target nodes.

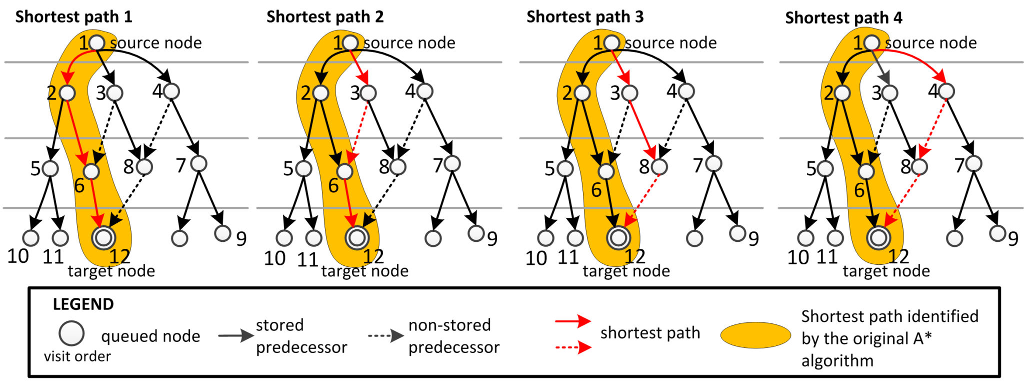


Figure 4.16: All shortest paths between the source node and the target node of the graph in the left-side of Figure 4.15, projected onto the exploration graph constructed by the $\mathbb{A}^{\star}$ algorithm (see the right-side of Figure 4.15). Note that all shortest paths are "connected" to the shortest path identified by the original $\mathbb{A}^{\star}$ algorithm.

The left figure of Figure 4.15 shows a directed graph where the distance of all edges is 1. The right side of Figure 4.15 shows an example of how the $\mathbb{A}^{\star}$ algorithm explores the nodes in the graph to identify a shortest path from the source node to the target node. Figure 4.16 shows the set of all shortest paths from the source node to the target node that can be identified from the node exploration graph shown in the right side of Figure 4.15. Note that all shortest paths in Figure 4.15 are connected to some nodes of the shortest path identified by the original $\mathbb{A}^{\star}$ algorithm.

For each queued node, the original $\mathbb{A}^{\star}$ algorithm only stores one predecessor node, i.e., the one that can be used to construct a shortest path to the node from the source node. If there are more than one predecessors that yield different paths with the same shortest distance, only one of them is stored. For example in Figure 4.15, node 3 is not stored as a predecessor of node 6 (i.e., the arc between node 3 and node 6 is dashed), because node 2 is already stored as the predecessor of node 6 and the shortest distance from the source node to node 6 either via node 2 or via node 3 is the same (i.e., distance of 2). To obtain more than one shortest path from the source node to the target node, for each queued node we store all of its predecessors that yield paths with the same shortest distance from the source node.

Furthermore, we do not stop iterating nodes in the priority queue after finding the first best candidate that is also a target node. Instead, we continue the iterations with the identified shortest distance value as a threshold. If the best candidate in the priority queue has a total shortest distance from the source node above the threshold, we stop the iteration. Algorithm 2 shows the pseudocode of the modified $\mathbb{A}^{\star}$ algorithm that finds all shortest paths between a source node and a target node.

Given a source node, a set of target nodes, and a directed graph with distance between nodes of the graph, we can obtain all shortest paths from the source node to the target nodes using Algorithm 2. Thus, given a trace and a process model, we can compute the set of all optimal alignments between them (see Theorem 4.4.2). Take for example a trace $\sigma_3 = \langle \mathit{add\ items}, \mathit{finalize}, \mathit{validate} \rangle$ and the process net shown in Figure 4.10. Figure 4.17 shows how an exploration graph is constructed in each iteration of Algorithm 2 (line 6-30) to compute all shortest paths between the source node and the target node

Algorithm 2: Pseudocode of an algorithm to obtain all shortest paths from a source node to a set of target nodes, assuming that the source node is not in the set of target nodes

```
initialize pqueue with the source node;
visitedNodesSet $\leftarrow \emptyset$;
solutionNodesSet $\leftarrow \emptyset$;
solutionFound $\leftarrow$ false;
distanceLim $\leftarrow +\infty$;
while pqueue is not empty do
    currNode $\leftarrow$ best candidate node in pqueue (node with the minimum total
    distance+underestimation distance to the nearest target node);
    if total distance + estimation of currNode $\leq$ distanceLim then
        if currNode $\in$ the set of all target nodes then
            solutionFound $\leftarrow$ true;
            distanceLim $\leftarrow$ shortest distance to reach currNode;
            solutionNodesSet $\leftarrow$ solutionNodesSet $\cup$ {currNode};
        end
        forall succNode $\in$ set of all successors of currNode do
            if succNode $\in$ visitedNodesSet then
                if (stored best distance to reach succNode) $>$ (current distance to
                reach succNode) then
                    replace the values of stored best predecessor and total distance
                    for succNode with the current ones;
                else if (stored best distance to reach succNode) $=$ (current distance to
                reach succNode) then
                    store the current predecessor together with the old predecessors;
                end
            else
                visitedNodesSet $\leftarrow$ visitedNodesSet $\cup$ {succNode};
                store predecessor and total distance to reach succNode;
                add succNode to pqueue;
            end
        end
    else
        break while;
    end
end
if (solutionFound = true) then
    forall solutionNode $\in$ solutionNodesSet do
        recursively iterate all predecessors of solutionNode until the source node to
        obtain all shortest paths;
    end
    return all shortest paths;
end
```

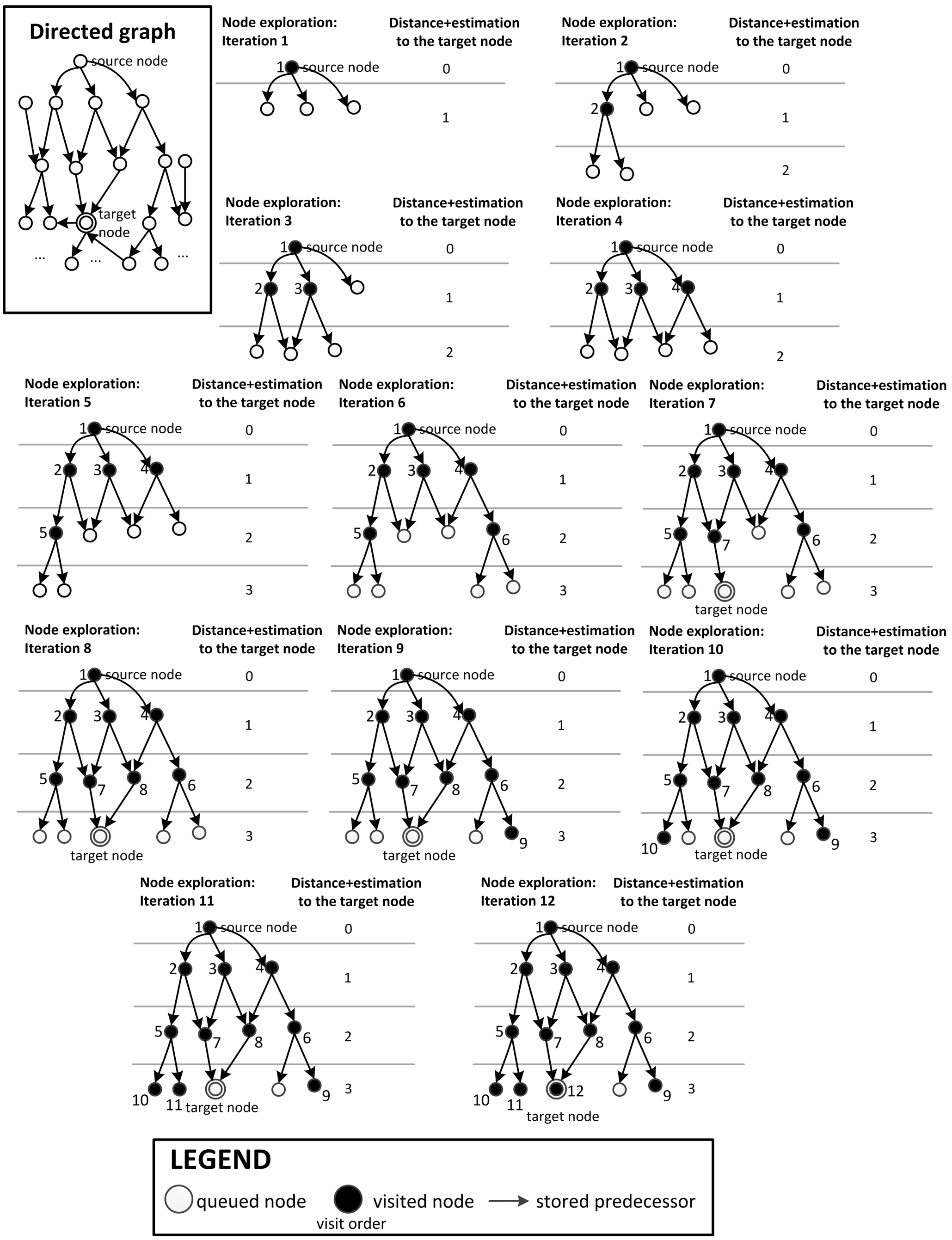

Directed graph
source node
target node
Node exploration: Iteration 1
Distance+estimation to the target node
Node exploration: Iteration 2
Node exploration: Iteration 3
Node exploration: Iteration 4
Node exploration: Iteration 5
Node exploration: Iteration 6
Node exploration: Iteration 7
Node exploration: Iteration 8
Node exploration: Iteration 9
Node exploration: Iteration 10
Node exploration: Iteration 11
Node exploration: Iteration 12
0
1
2
3
LEGEND
queued node
visited node
visit order
stored predecessor

Figure 4.17: Top Left: a directed graph where all arcs have distance 1, and **Other:** Illustration on how the nodes of the graph are explored in each iteration of line 6-30 of Algorithm 2 using a permissible underestimation function that always returns 0 for all nodes.
]

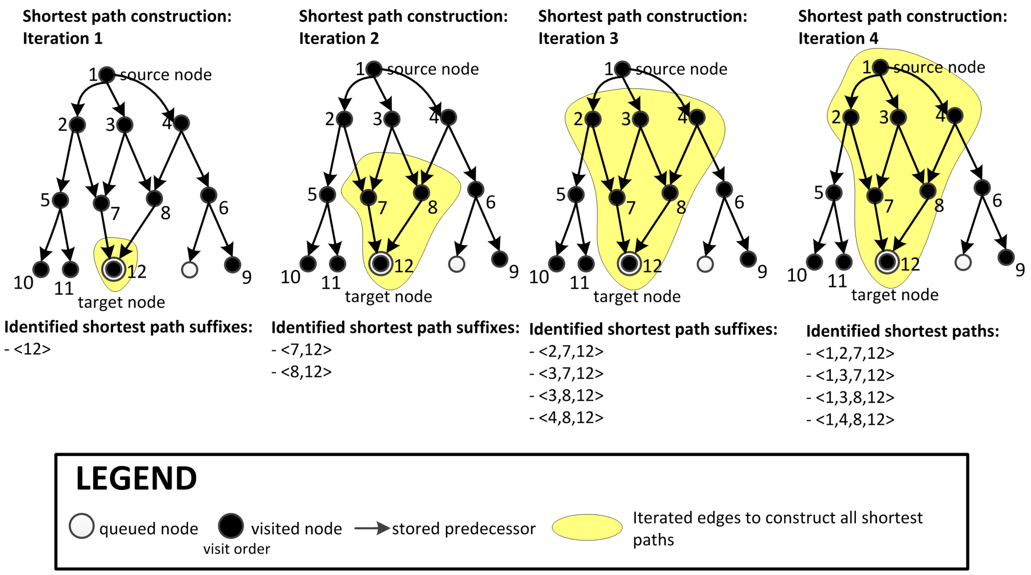

Shortest path construction: Iteration 1
source node
target node
Identified shortest path suffixes:
- <12>
Shortest path construction: Iteration 2
Identified shortest path suffixes:
- <7,12>
- <8,12>
Shortest path construction: Iteration 3
Identified shortest path suffixes:
- <2,7,12>
- <3,7,12>
- <3,8,12>
- <4,8,12>
Shortest path construction: Iteration 4
Identified shortest paths:
- <1,2,7,12>
- <1,3,7,12>
- <1,3,8,12>
- <1,4,8,12>
LEGEND
queued node
visited node
visit order
stored predecessor
Iterated edges to construct all shortest paths

Figure 4.18: Illustration on how all shortest paths are constructed from the final exploration graph obtained in Figure 4.17 using Algorithm 2. Stored predecessors are iterated backward from the target node.

of the directed graph shown in the left side of Figure 4.15. Figure 4.18 shows a step-by-step construction of all shortest paths from the obtained exploration graph (i.e., line 33 of Algorithm 2). The set of constructed shortest paths is the same as the one shown in Figure 4.16.

Using Algorithm 2 to compute all shortest path between the trace and the model, we obtain the set of all optimal alignments between σ_3 and the net as shown in Figure 4.19. Note that some alignments are similar to others. For example, γ_2 can be obtained from γ_1 by swapping the move on model $(\gg, t_4)$ on column 3 with the move on model $(\gg, t_5)$ on column 4. γ_4 can be obtained from γ_3 by swapping the same pair of columns. Transitions t_4 and t_5 can be fired in any order according to the net. Similarly, γ_5, γ_6, and γ_7 have exactly the same set of movements with different ordering between two moves on log $(\mathit{finalize}, \gg)$ and $(\mathit{validate}, \gg)$ and a move on model $(\gg, t_9)$. In some cases, instead of having all optimal alignments, having some representatives of all optimal alignments may provide better deviation diagnostics.

Therefore, we introduce the notion of *representatives* of all optimal alignments. Given a trace and a process net, the approach presented in Section 4.4 computes an optimal alignment that *represents* all optimal alignments between the trace and the net. Algorithm 2 computes the set of all optimal alignments between the trace and the net, thus each alignment in the set represents itself. Given a set of optimal alignments, the idea is to group alignments that have similar characteristics. For example, two alignments where each of them can be constructed by reordering the elements of the other should be grouped together, e.g., alignment γ_1 and γ_2 in Figure 4.19.

We take a pragmatic approach to group all optimal alignments between a trace and a process net without introducing too much computation overhead. We use the exploration graphs constructed as byproducts of Algorithm 2. Take for example the exploration graph shown in Figure 4.18 and all four shortest paths from the source node to the target node

$\gamma_1 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pay*	*pack*	*validate*	*deliver*
t_1	t_3	t_4	t_5	t_6	t_7

$\gamma_2 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pack*	*pay*	*validate*	*deliver*
t_1	t_3	t_5	t_4	t_6	t_7

$\gamma_3 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pay*	*pack*	*validate*	*cancel*
t_1	t_3	t_4	t_5	t_6	t_8

$\gamma_4 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pack*	*pay*	*validate*	*cancel*
t_1	t_3	t_5	t_4	t_6	t_8

$\gamma_5 =$

add items	$\gg$	*finalize*	*validate*
add items	*cancel*		
t_1	t_9	$\gg$	$\gg$

$\gamma_6 =$

add items	*finalize*	$\gg$	*validate*
add items		*cancel*	
t_1	$\gg$	t_9	$\gg$

$\gamma_7 =$

add items	*finalize*	*validate*	$\gg$
add items			*cancel*
t_1	$\gg$	$\gg$	t_9

Figure 4.19: All optimal alignments between trace $\sigma_3 = \langle \mathit{add\ items}, \mathit{finalize}, \mathit{validate} \rangle$ and the process net in Figure 4.10.

identified from the graph shown in Figure 4.16. Figure 4.20 shows some possible ways to group shortest paths from the graph shown in Figure 4.16. All edges in the yellow area are parts of some shortest paths from the source node (with visit order 1) to the target node (with visit order 12). We choose some representatives of all shortest paths by first choosing a set of nodes to "knot" some shortest paths. These nodes are called *knot nodes*. Two shortest paths in an exploration graph are grouped together by a knot node if they contain an edge that points to the knot. For example, the leftmost figure in Figure 4.20 shows a situation where the target node is chosen as the only knot node. Since all shortest paths reach the target node, i.e., they contains an edge that points to the knot node, all shortest paths in the figure are grouped as one group. In the other extreme, the rightmost figure shows a situation where the source node is chosen as the only knot node. Since all edges go out from the source node, there are no groupings on shortest paths, i.e., each shortest path belong to a group.

Given an exploration graph, the set of knot nodes for the graph can be chosen randomly. However, to maximize the similarity between shortest paths that are grouped together without introducing too much computation overhead, we choose the set of knot nodes based on the number of edges that need to be iterated from the target node. For example, the knot nodes in the second-left figure of Figure 4.20 are the nodes reachable

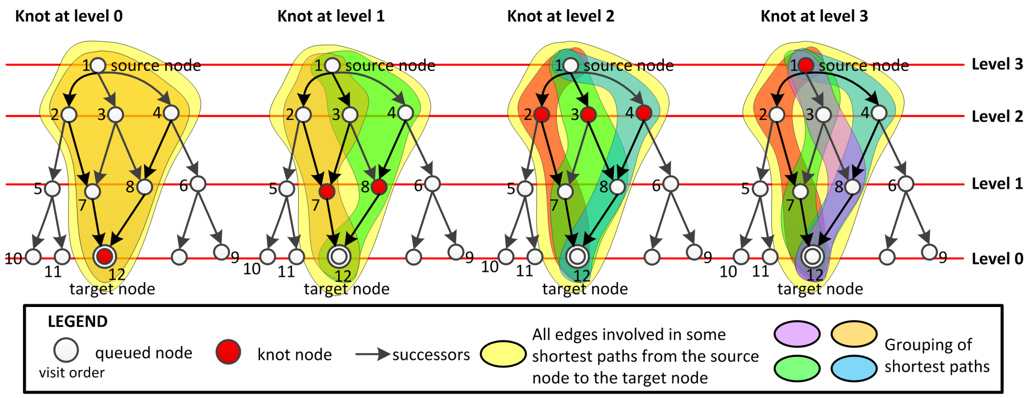


Figure 4.20: The final exploration graph and identified shortest paths taken from Figure 4.18. Shortest paths are grouped by choosing different levels of knot nodes. The number of groups increases as the selected level of knot nodes increases.

$\gamma_1 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pay*	*pack*	*validate*	*deliver*
t_1	t_3	t_4	t_5	t_6	t_7

$\gamma_2 =$

add items	*finalize*	$\gg$	$\gg$	*validate*	$\gg$
add items	*finalize*	*pay*	*pack*	*validate*	*cancel*
t_1	t_3	t_4	t_5	t_6	t_8

$\gamma_3 =$

add items	$\gg$	*finalize*	*validate*
add items	*cancel*		
t_1	t_9	$\gg$	$\gg$

Figure 4.21: Representatives of all optimal alignments between trace $\langle add\ items, finalize, validate\rangle$ and the process net in Figure 4.10 using knot nodes of level 1.

after 1 backward iteration from the target node. Thus, we say that the shortest paths are knotted at level 1. The knot nodes in the third-left figure of Figure 4.20 are reachable after 2 backward iteration from the target node, thus the shortest paths are knotted at level 2. We use knot nodes at different level to group some optimal alignments together and take one random representative from each group. Notice that the number of grouped shortest paths decreases as the level of knot nodes increases.

Given an exploration graph and a set of shortest paths identified from the graph, knot level offers a “slider” mechanism to group shortest paths with some degree of similarity. Figure 4.21 shows the set of representative optimal alignments obtained from the example shown in Figure 4.19 using knot nodes of level 1. Notice that the each representative is unique and cannot be reproduced by simply swapping some movements of other representatives.

Note that the strategy to select knot nodes based on the number of iterated arcs from the target node does not guarantee that a shortest path belongs to just one group, i.e., a shortest path may belong to multiple groups. One can also think of other strategies to choose knot nodes. For example, instead of using the target node as a reference one can also use the source node as reference and select knot nodes based on certain number of arcs need to be iterated from the source node. In this way, two shortest paths are grouped together if they are started with the same sequence of edges. We can use the same knot node strategy to obtain representatives of optimal prefix alignments.

4.7 Experiments

Given a trace and a process net, Section 4.4 to Section 4.6 explain various ways to compute optimal (prefix) alignments between them and their representatives. In this section, we show some experimental results to evaluate the proposed approaches.

We implemented the approach in ProM [198] and used the lp_solve tool as the ILP solver in our implementation[2]. Two sets of experiments were performed. In the first set of experiments, we compared the proposed approach with existing approaches using two similar artificial process nets both having a real-life complexity. The second set of the experiments was performed to show the applicability of the approach to handle real life models and logs. We used case studies from the CoSeLog project involving several municipalities in the Netherlands. The first set of experiments is explained in Section 4.7.1, while the case study results are explained in Section 4.7.2.

4.7.1 Artificial Logs and Models

The goal of this set of experiments is to evaluate the robustness of the proposed approach to compute alignments between traces and complex process models. The approaches proposed in Section 4.5 and Section 4.6 are two extensions of the basic approach proposed in Section 4.4. Thus, in this section we focus on the evaluation of the approach introduced in Section 4.4. In particular, we focus on two aspects: *memory efficiency* and *computation time*.

We compared the two approaches proposed in Section 4.4: the one with naive underestimation function (see Theorem 4.4.4) and the one with a more precise underestimation function using ILP (see Theorem 4.4.9). In addition, we compared both approaches with the tree-based state space exploration approach proposed by Cook and Wolf [39] since this approach is most related to our work. We created two artificial process nets and a set of logs generated from both nets. Both nets loosely describe the process of applying for a building permit in a municipality in the Netherlands. One process net is a Petri net (see Figure 4.22), and the other is a reset/inhibitor net (see Figure 4.23). For the sake of readability, some parts of the net are grouped into subprocesses. Both process nets have the same subprocess of *regular permit check* as shown in Figure 4.24. Furthermore, both of them contain invisible/duplicate transitions and complex control-flow patterns (e.g., OR-splits, loops, and choices).

We generated perfectly fitting traces from each net (traces that can be perfectly replayed) with various lengths between 20 to 69 activities per trace, and then introduced noise by randomly removing and/or inserting activities. Then, we constructed an optimal alignment for each trace and its net and recorded the number of queued states (i.e., the states that are actually visited and others that are considered as candidates to be visited) needed to construct it.

[2]see `http://sourceforge.net/projects/lpsolve`

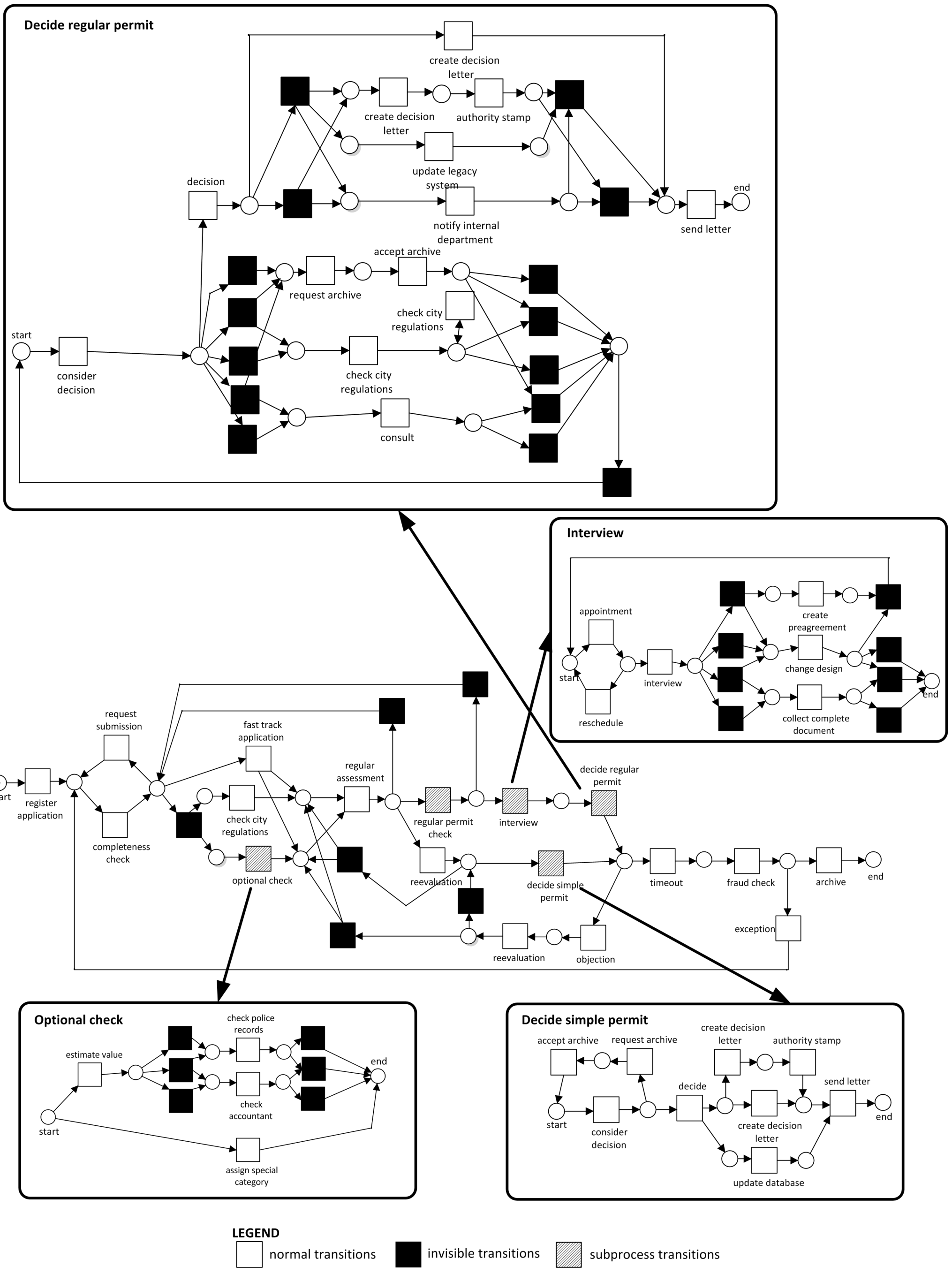


Figure 4.22: The Petri net used in the experiments and details of some of its subprocesses.

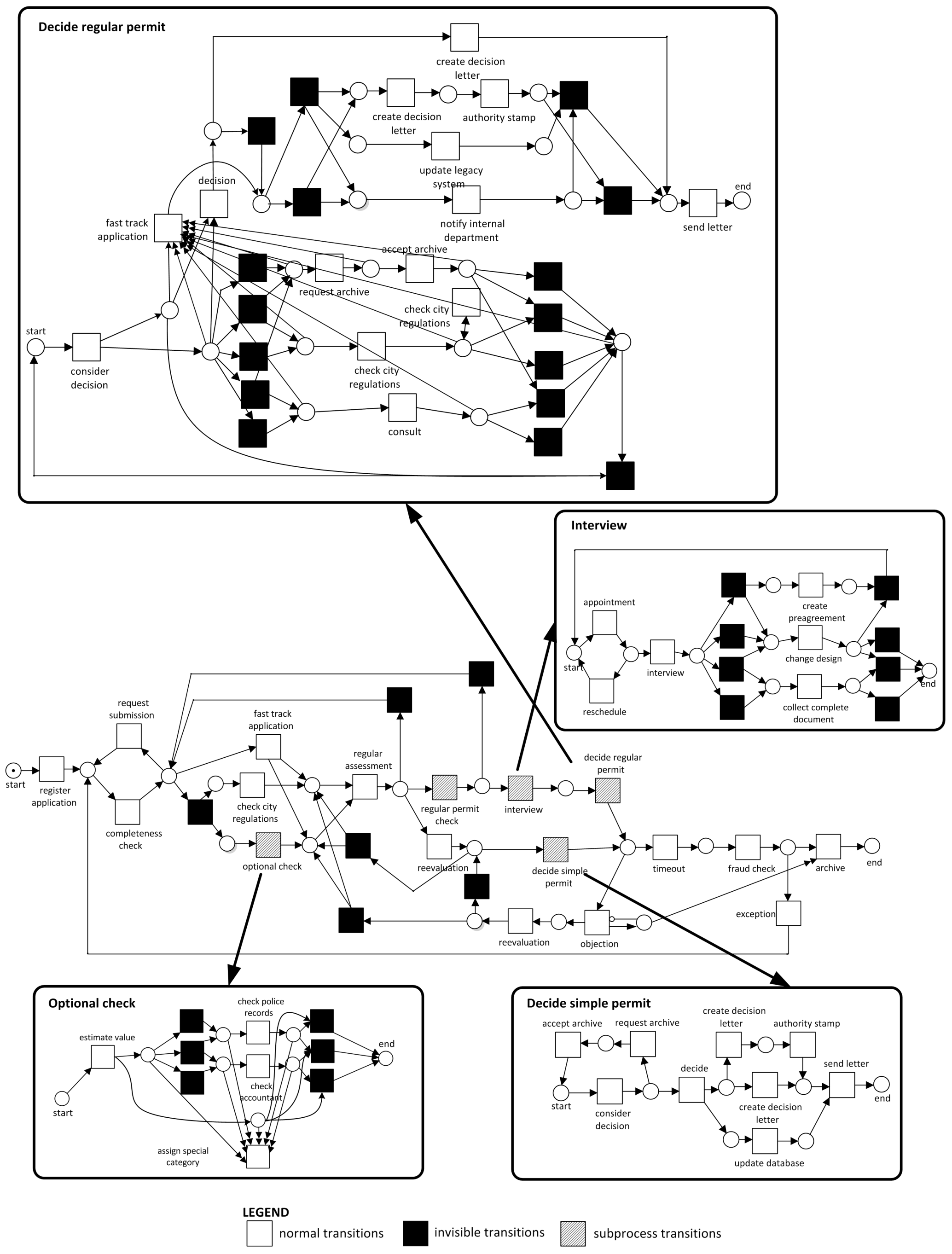


Figure 4.23: The reset/inhibitor net used in the experiments and details of some of its subprocesses.

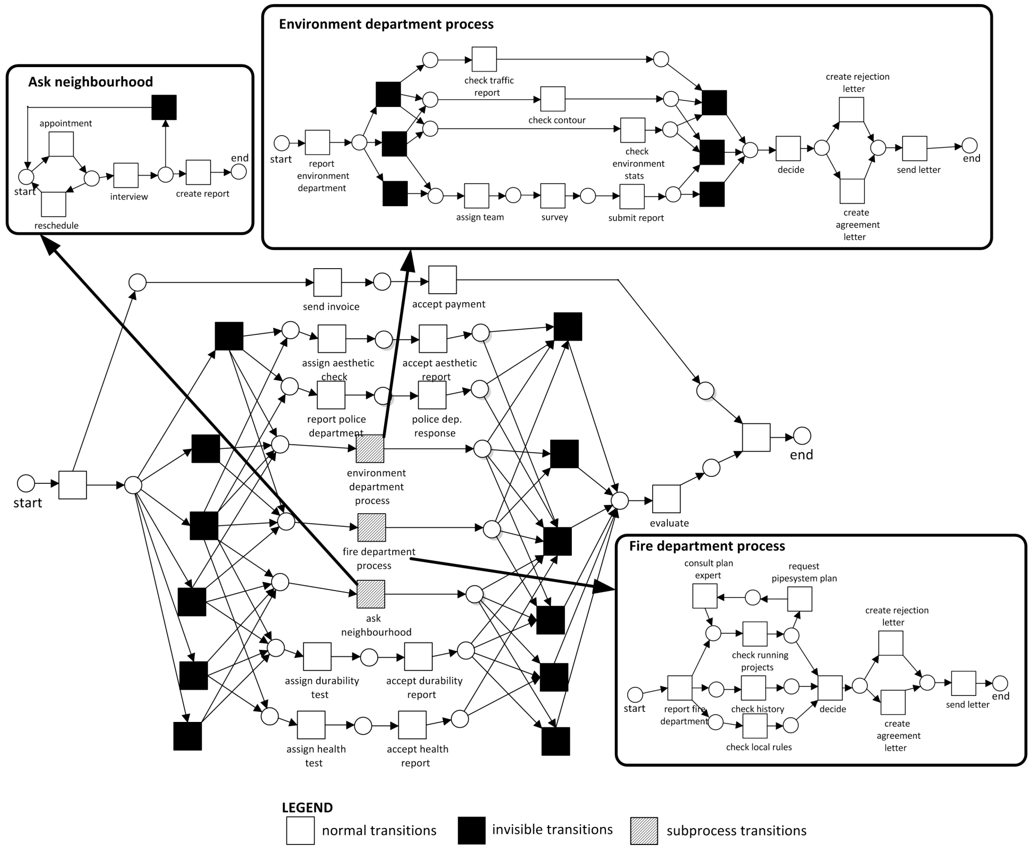


Figure 4.24: The sub-process of transition *regular permit check* in both Figure 4.22 and Figure 4.23.

We use the standard cost function for all of the experiments. However, since the $\mathbb{A}^\star$ algorithm requires positive non-zero cost (see the discussion about the $\mathbb{A}^\star$ algorithm in the end of Section 4.4.1), we add all costs by a negligibly small value $\epsilon = 0.001$. For benchmarking, we did the same set of experiments with the tree state-space-based approach proposed in [39]. We use a computer with Intel Xeon 2.66 GHz processor and use 1 GB of Java Virtual Memory. The results are shown in Figure 4.25 to Figure 4.27. Each dot in the figures is based on the average of performing the same experiment 30 times, each with a different log consisting of 100 traces. The vertical bars indicate the corresponding 95% confidence interval. Note that in all figures, the y-axis is shown using a logarithmic scale.

Figure 4.25 shows that the number of explored states to construct alignments increases as the length of traces and noise level increases. The figure shows that the approach with ILP computation explores much fewer states to construct alignments than other approaches in all cases. Furthermore, it is less sensitive to noise than other approaches. The permissible underestimation functions defined in Theorem 4.4.9 manage to estimate the cost such that only relevant states that actually lead to solutions are explored. In cases where both the approach without ILP and the tree-state-space based exploration need to choose which transition to fire for an OR-split/join pattern, both of them use random selection that may lead to the exploration of many irrelevant states

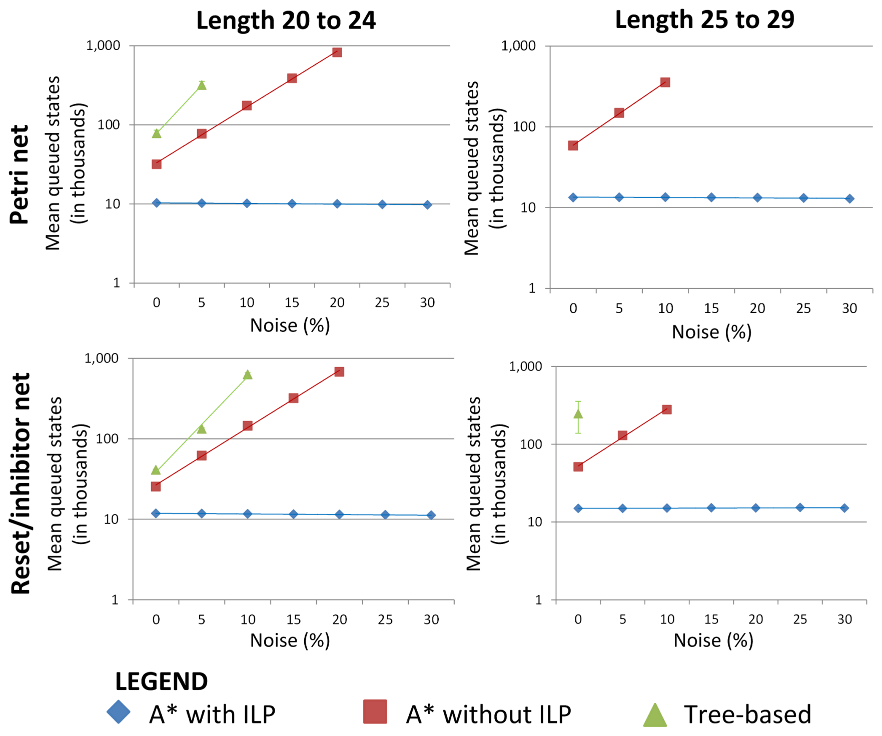


Figure 4.25: The number of explored states to construct optimal alignments for the Petri net in Figure 4.22 and the reset/inhibitor net in Figure 4.23. Each dot in the figures is based on the average of performing the same experiment 30 times with different noisy logs where each log consists of 100 traces. Missing values are due to out-of-memory problems of tree-based and $\mathbb{A}^\star$ without ILP. Clearly, the $\mathbb{A}^\star$ with ILP outperforms the $\mathbb{A}^\star$ without ILP and the tree-based state exploration.

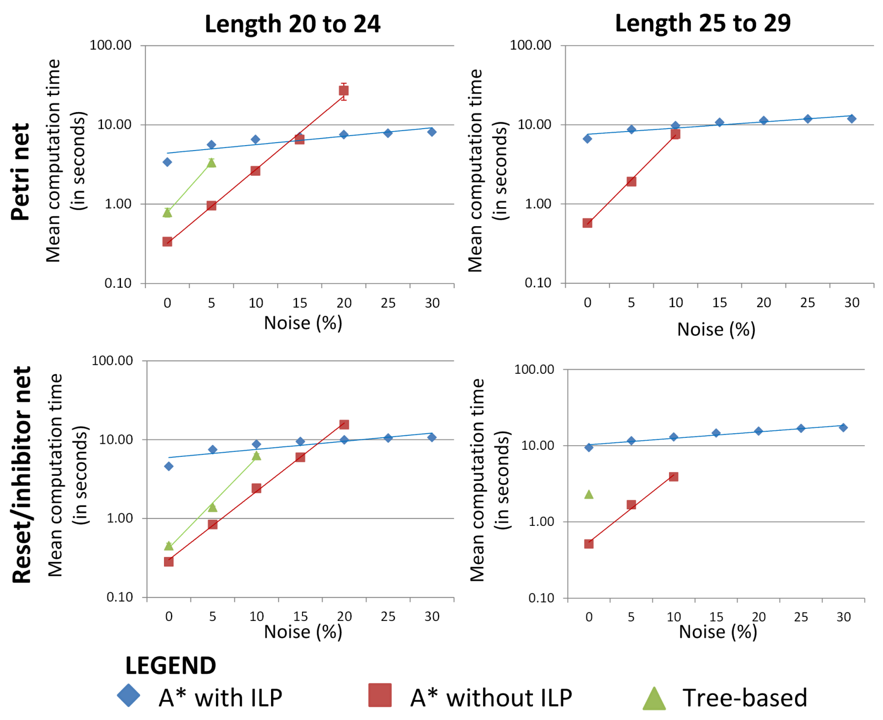


Figure 4.26: The computation time required to construct optimal alignments for the Petri net in Figure 4.22 and the reset/inhibitor net in Figure 4.23. Each dot is based on the average of performing the same experiment 30 times with different noisy logs where each log consists of 100 traces. Missing values are due to out-of-memory problems of tree-based and $\mathbb{A}^*$ without ILP. For lower noise levels, the $\mathbb{A}^*$ without ILP is the fastest. For higher noise levels, the $\mathbb{A}^*$ with ILP is faster and often the only one to terminate.

before they finally explore the correct ones. We can also see that the estimation function significantly reduces the number of states that need to be investigated in cases where there is noise.

Figure 4.25 also shows that only the ILP-based approach managed to compute optimal alignments in all experiments. Other approaches have out-of-memory problems when dealing with either large or noisy logs. For example, in the experiment with traces of length between 20 and 24 (see Figure 4.25, top-left), the tree-based state space approach [39] only managed to compute optimal alignments until the noise level reaches 5%. Above 5% noise level, there are too many states that need to be explored by the approach such that out-of-memory problems occurred. The non-ILP approach performs better than the tree-based state space approach, but it can only provide results up to noise level 20% before out of memory problem occurred. Note that other than storing queued states, all approaches need to store the structure of constructed exploration graphs in memory.

Figure 4.26 shows the computation time needed to construct optimal alignments using different approaches. As shown in Figure 4.26, the approach with ILP requires more computation time than the others if there is no noise. The overhead of computing the ILP per visited state does not pay off if there are no or just few deviations. However, in cases where noise exists and traces are long, the approach without ILP explores significantly more states than the one with ILP, such that its total computation time is higher. See for example the experiments involving a Petri net without reset and inhibitor arcs

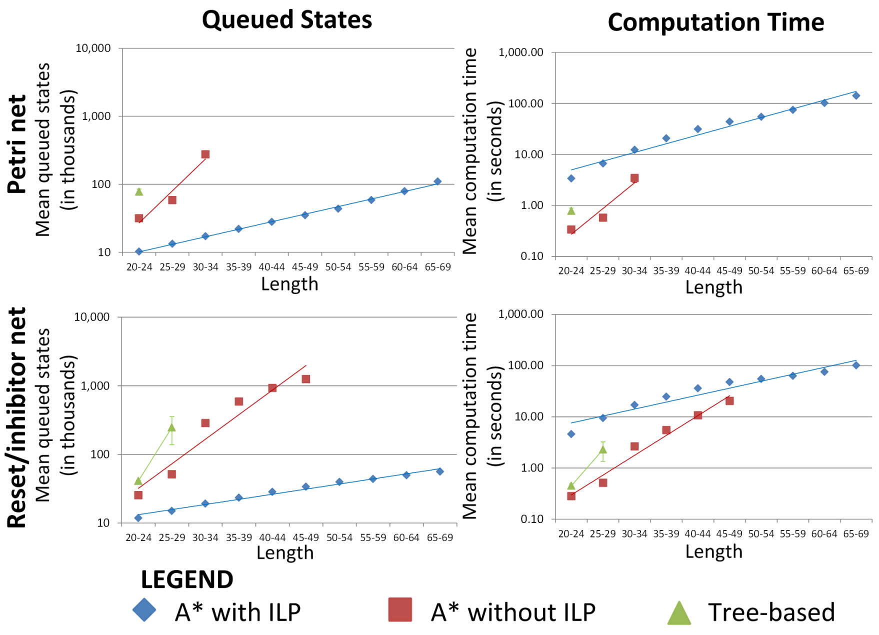


Figure 4.27: The computation time and the number of explored states to construct optimal alignments for the Petri net in Figure 4.22 and the reset/inhibitor net in Figure 4.23. Each dot is based on the average of performing the same experiment 30 times with different perfectly fitting logs where each log consists of 100 traces. Missing values are due to out-of-memory problems of tree-based and $\mathbb{A}^\star$ without ILP. $\mathbb{A}^\star$ with ILP is the only algorithm that manages to provide optimal alignments for traces with high noise level.

and log with traces of lengths between 20 to 24 events in top-left of Figure 4.26. When noise level reaches 20%, the ILP approach has lower computation time than the one without ILP. Similarly, the experiment with the reset/inhibitor net and traces of same length shows the same result when noise level reaches 20% (see Figure 4.26, bottom-left). Moreover, for larger noise levels the tree-based and $\mathbb{A}^\star$ without ILP are unable to compute alignments due to out-of-memory problems.

Figure 4.27 shows experimental results using the same models and logs with perfectly fitting traces of various lengths. As shown in the figure, only the ILP approach managed to provide optimal alignments for all experiment logs while the others fail at logs with long traces due to out of memory problems. This underlines the importance of having a memory-efficient alignment approach.

As mentioned in Section 4.5 and Section 4.6, the approaches to compute optimal prefix alignments, all optimal (prefix) alignments, and all representatives are all based on the approach of computing one optimal alignment per trace in Section 4.4. Given a trace and a Petri net, the approach to compute one optimal prefix alignment for the trace and the net is similar to the approach to compute one optimal alignments using the permissible underestimation function in Theorem 4.4.4. Therefore, we believe that the outcome on computation time and memory use of the computation of prefix optimal alignment resembles the results of using the $\mathbb{A}^\star$ algorithm without ILP in this section.

The complexity of computing all optimal (prefix) alignments between the trace and the net is obviously higher than computing just one optimal (prefix) alignment between them. The computation of all optimal (prefix) alignments requires an extra iteration on

Table 4.1: Real-life logs and models used in experiments

Num	Log			Petri net		Net Features				
	Name	**#Traces**	**#Events**	**T**	**P**	**Invisible**	**Duplicate**	**Parallel**	**Choice**	**Loop**
1	LM01	3,181	20,491	12	15	yes	-	yes	-	-
2	LM02	1,861	15,708	19	16	yes	-	-	yes	yes
3	LM03	10,271	85,548	21	24	-	-	yes	-	-
4	LM04	4,852	29,737	27	16	yes	-	-	yes	-
5	LM05	25,846	141,755	24	14	yes	-	-	yes	-
6	BouwV	714	9,116	33	23	yes	-	-	yes	yes
7	Bouw1	139	3,364	34	33	yes	yes	yes	yes	yes
8	Bouw2	121	2,247	30	28	yes	-	yes	yes	yes
9	Bouw3	94	913	31	31	yes	-	yes	yes	yes
10	Bouw4	109	2,331	31	31	yes	-	yes	yes	-

priority queue and also extra steps to reconstruct alignments from all identified paths. In the worst case, the extra computation that one needs to performed to obtain all optimal (prefix) alignments is exponential in the size of the shortest path. Therefore, we believe that the both computation time and memory requirement of the approach to compute all optimal (prefix) alignments is much more than the ones required to compute one optimal (prefix) alignment. In Section 4.7.2, we show the experimental results using real life logs and models.

4.7.2 Real Life Cases

Alignments are the starting point for various types of analysis based on both observed and modeled behavior. To show that the approach shows various insights and robust to logs and models with real-life complexity, we took several real-life logs and models as case study. The logs and models were taken from Dutch municipalities, mostly the ones involved in the CoSeLog project [26]. Most logs and models are related to building permit application handling, and some others are related to objection handling of building permit decision. Details about the logs and the models are shown in Table 4.1.

First, we compared the deviation diagnostics and time required to compute 1 optimal (prefix) alignment per trace using all pairs of logs and models. Figure 4.28 shows the comparison results. As expected, optimal alignments show higher average number of deviations per trace than prefix alignment because uncompleted traces are penalized. In the experiments with logs/models “LM03” and “BouwV”, the difference between the average deviation cost per trace provided by optimal alignment and optimal prefix alignment are much higher than others. This is an indication that in the two logs, many of the traces are incomplete compared to other logs. The right chart of Figure 4.28 shows that the computation time of both approach are relatively low (lower than 100 ms per trace for all logs and models). If we only take into account the time required to replay all unique traces in the logs, the total computation time for each pair of log and model in the experiments is always below 9 seconds (not shown in the graph). This shows that the approaches to compute one optimal (prefix) alignment per trace in Section 4.4 and Section 4.5 are robust to handle logs and models with real-life complexity.

Figure 4.29 shows diagnostics provided by optimal (prefix) alignments for the same set of traces in log “LM03”. Notice that there are more moves on model identified by the

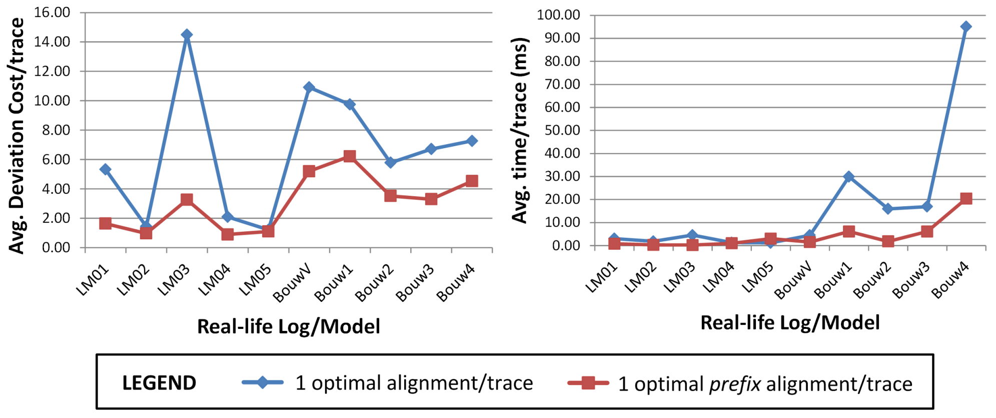


Figure 4.28: Comparison of results obtained from experiments of computing a single (prefix) optimal alignment per trace using real life logs/models. **Left:** The optimal alignment approach yields higher deviation cost as it penalizes non-completion. **Right:** Computing optimal alignments (with ILP) requires more time than optimal prefix alignments.

approach to compute optimal alignments because the traces are not properly terminated. Such a proper termination is not required for optimal prefix alignments, hence it is not penalized. The diagnostics shown in Figure 4.29 represent 1,854 + 497 + 479 + 362 = 3,192 traces. There are 10,271 traces in the log. Hence, Figure 4.29 shows more than 30% of the total number of traces in the log. This implies that at least 30% of traces in the log need to be filtered out if the log is about to be used for analysis that require completed traces, e.g., the average throughput time of cases.

We also computed all optimal alignments and representatives of all optimal alignments for all pairs of logs/models. In all experiments, we chose the set of knot nodes of level 1 to get representatives of all optimal alignments. We recorded the maximum and average number of all optimal alignments per trace. Furthermore, we recorded the number of representatives, as well as the number of optimal alignments they represent. The results of the experiments are shown in Table 4.2.

In few of the set of all experiments, out-of-memory exceptions occurred when computing all optimal alignments between traces in the event logs and models. The reason for these out-of-memory problems is simply because there are too many optimal alignments between the traces in the logs and the models. For example, the experiment with log/model “LM03” shows that the maximum number of all optimal alignments between a trace in the log and the model is more than 32 million (i.e., a representative in the log represents more than 32 million optimal alignments). Other pairs of logs and models that have the same exceptions (“Bouw1” and “Bouw4”) also have a high maximum number of represented optimal alignments per trace. In such cases, constructing all optimal alignments is obviously memory-demanding.

In all experiments, we managed to obtain all representatives of all optimal alignments. The number of obtained representatives per trace typically varies between 1 and 2. This shows that for real life logs and models, variations between optimal alignments rarely occur at the end of the alignments. In fact, the variations are spread along the alignments. If we only compute all representatives, the maximum required computation time is below 21 seconds, which is acceptable for real-life use case.

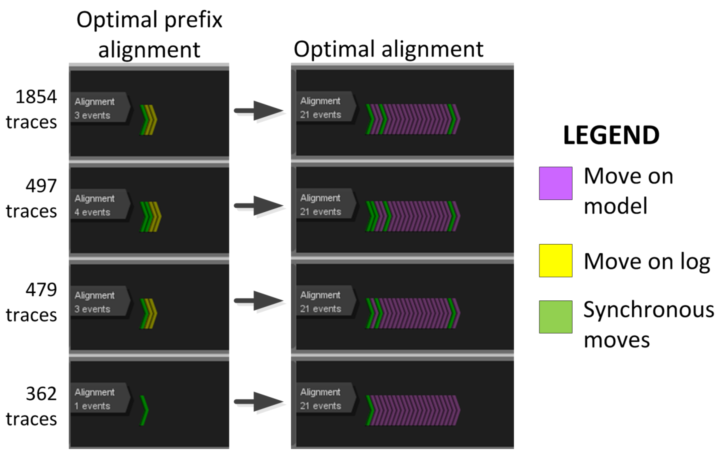


Figure 4.29: Screenshots of the implemented approach in ProM 6, showing a comparison between deviation diagnostics provided by one optimal prefix alignment (left) and one optimal alignment (right) in the experiment with log/model “LM03”.

Table 4.2: Results of Computing More than One Optimal Alignments/Trace in Real-life Logs/Models

Num	**Log**	**Representative Alignments, knot level-1**				**All Optimal Alignments**		
		#Representatives per Trace		**Max. #Represented Opt. Alignments**	**Total Time (seconds)**	**#Alignments per Trace**		**Total Time (seconds)**
		max	**avg.**			**max**	**avg.**	
1	LM01	2	1.03	4,272	$<$ 1 sec	6,281	158.00	1 sec
2	LM02	2	1.15	1,716	$<$ 1 sec	3,003	4.99	$<$ 1 sec
3	LM03	2	1.17	32,686,880	8 sec	**o/m**	**o/m**	**o/m**
4	LM04	2	1.00	268	$<$ 1 sec	328	4.30	$<$ 1 sec
5	LM05	2	1.00	316,029	1 sec	316,029	50	$<$ 60 sec
6	BouwV	2	1	15,016	$<$ 1 sec	24,025	105	$<$ 1 sec
7	Bouw1	2	1	22,118,400	$<$ 7 sec	**o/m**	**o/m**	**o/m**
8	Bouw2	2	1	1,216	$<$ 1 sec	1,216	38	$<$ 1 sec
9	Bouw3	2	1	8,829	$<$ 1 sec	8,829	166	$<$ 1 sec
10	Bouw4	2	1	67,376,336	$<$ 21 sec	**o/m**	**o/m**	**o/m**

o/m : Out-of-memory exception

4.8 Conclusions

In situations where process executions are not enforced by systems, deviations between the behavior of an organization and the models used to describe the ideal behavior of the organization occur frequently. The analysis of business processes based on observed behavior has proven to be a complex problem in such situations, because analysis techniques typically have difficulties relating the observed behavior to the modeled behavior. In this chapter, we showed how to compute various types of alignments between observed behavior in the form of event logs and process models in form of Petri nets or reset/inhibitor nets.

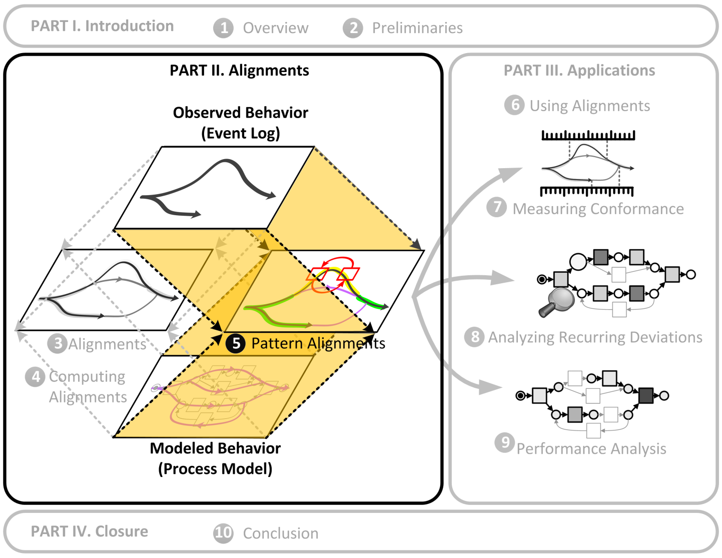
Aligning traces with Petri nets or reset/inhibitor nets is a complex problem which we addressed by translating this problem into a shortest path problem on a (possibly infinite) graph. We showed that a shortest path can always be found using an $\mathbb{A}^\star$ based algorithm with a permissible underestimation function. The techniques presented in this chapter use the relation between the structure of the Petri net and its potential behavior optimally to improve computation efficiency. We exploited the marking equation in our estimation function. The experiments using real-life size logs and models showed that memory use is reduced significantly in all cases where the marking equation is exploited. In situations where no deviations occur, the computation time overhead caused by computing the marking equation slightly increases the overall computation time. However, in cases where process models are complex and deviations are more frequent, the use of the marking equation leads to a reduction in computation time next to the memory reduction. In fact, without using the marking equation, out-of-memory problems occur already for moderate-sized models and logs. By exploiting the marking equation we can analyze processes which are an order of magnitude larger. Furthermore, we showed using several study cases that the approaches are robust to logs and models with real-life complexity, and alignments in general provide valuable insights into process executions.

In order to make our approach applicable to real-life languages such as BPMN, EPCs, etc, we extended our techniques to Petri nets with reset and inhibitor arcs. This way we can deal with advanced workflow patterns, such as cancellation, priorities, OR-joins, and timeouts more easily. Languages like BPMN, EPCs, UML Activity Diagrams, and YAWL can easily be translated to Petri nets with reset and inhibitor arcs while retaining precise semantics. Our experiments show that the introduction of these arcs does not lead to a significant increase in memory usage or computation time.

Furthermore, given a trace and a process model, we extend the approach to compute an optimal prefix alignment, all optimal (prefix) alignments, and a set of representatives of all optimal (prefix) alignments between them. This way, we provide more types of deviation diagnostics between the trace and the model. Although the extensions are computationally more expensive than the original approach to compute one optimal alignment per trace, real-life experiments show that these approaches are applicable to logs/models of low to medium complexity and that they provide complementary diagnostics to the one provided by the one optimal alignment per trace approach.

Chapter 5

Extension to High-Level Deviations



5.1 Introduction

Alignments provide a way to measure the severity of deviations and explicit diagnostics on where the deviations are. Given a trace, a process model, and a likelihood cost function of movements, deviations are explicitly shown by moves on model and moves on log of an optimal alignment between the trace and the model with respect to the cost function. Furthermore, the total cost of the alignment indicates the likelihood of the deviations. However, diagnostics for log/model moves tend to be too low-level. Often measures and diagnostics at a higher-level of granularity are desired [53, 162]. For example, high-level management in organizations may have more interest in knowing whether two activities are often swapped or whether an activity is often performed instead of some other activities in the trace.

Given a trace and a process model, low-level deviations in the form of moves on

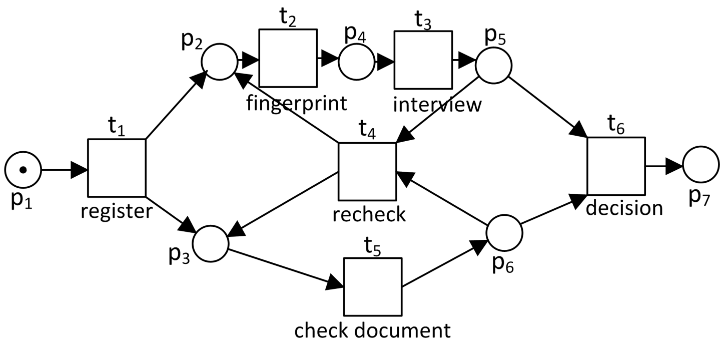


Figure 5.1: A visa application handling process.

register	≫	*interview*	≫	*fingerprint*	*recheck*	*decision*
register t_1	*fingerprint* t_2	*interview* t_3	*check document* t_5	≫	≫	*decision* t_6

Figure 5.2: An optimal alignment between trace $\sigma_1 = \langle register, interview, fingerprint, recheck, decision\rangle$ and the model in Figure 5.1, showing low-level deviations.

model/log are the "building blocks" for the higher-level deviations between them. The trace does not perfectly fit the model if and only if there is either a move on model or a move on log in an optimal alignment between them. High-level deviations may not be easily identified from low-level deviations. Take for example the process of a visa application handling in Figure 5.1. The process starts when an applicant files a visa application and registers himself in a visa center (register). While the documents of the application are checked by an officer (check document), another officer collects the applicant's fingerprint (fingerprint) using a machine and then performs an interview (interview). If suspicious information is found in the application, a manager may decide to recheck the applicant (recheck). Rechecking activity may include checking all documents of the applicant. The process terminates after the manager makes a decision on the application (decision).

Suppose that in an instance of the process, the fingerprint machine has some problems and therefore the fingerprint activity is frequently swapped with interview. Furthermore, a manager performs the recheck procedure instead of check documents because an officer that should perform the latter activity is absent. Trace $\sigma_1 = \langle$*register*, *interview*, *fingerprint*, *recheck*, *decision*$\rangle$ shows a trace where both high-level deviations occur. An optimal alignment between the trace and the process model is shown in Figure 5.2. Figure 5.2 explicitly shows that there are four low-level deviations: two moves on model and two moves on log. Identifying that fingerprint is swapped with interview and recheck replaces check documents in this alignment is not trivial. The combination of move on model $(\gg, t_2)$ and move on log $(fingerprint, \gg)$ indicates that the activity fingerprint is misplaced in the trace, but it does not explicitly show that it is swapped with another activity in the trace, i.e., interview. Similarly, replacement of check documents with recheck cannot be easily inferred by pairing move on model $(\gg, t_5)$ and move on log $(recheck, \gg)$, as $(\gg, t_5)$ can also be paired with another movement $(fingerprint, \gg)$ that yields a different conclusion.

In this chapter, we extend the approach of computing optimal alignment in Chapter 4 to *check whether high-level deviations occur* in the trace. Section 5.2 provides a detailed explanation on an approach to check high-level deviations using alignments. Section 5.3 shows how the approach can be used to identify the possible root causes of deviations between the trace and the model. Related work is discussed in Section 5.4. Experiment results are provided in Section 5.5, and Section 5.6 concludes the chapter.

5.2 High-level Deviation Patterns

In this chapter, we only consider high-level deviations that are related to control-flow. *We explicitly model high-level deviations as patterns of low-level deviations*. To avoid ambiguity, we use Petri nets to express the deviation patterns. A *deviation pattern* is a Petri net fragment that represents a high-level deviation explicitly. Therefore, a high-level deviation pattern only allows the deviating behavior it represents and nothing else. Given a trace and a Petri net, a deviation pattern is first appended to the net in order to check the occurrences of the high-level deviation represented by the pattern in the trace. A deviation pattern that is appended to a net is an *instance* of the pattern in the net. If the trace contains a deviation that is represented by the pattern then an optimal alignment between the trace and the appended net should contain synchronous moves of transitions that belong to the appended pattern.

Some transitions in a deviation pattern are marked as input/output borders. Given a Petri net to be appended with a deviation pattern, all input/output borders in the pattern have the same input/output arcs as some transitions in the net. Hence, these borders "glue" all instances of the pattern to the original Petri net. Some transitions in a deviation pattern may have the same label. The initial marking and final marking of a deviation pattern are the empty marking [].

In this section, we provide some examples of deviation patterns, intuition behind each of them, and explanation on how to use them in order to identify high-level deviations. Without aiming to be complete, in this section we discuss the three deviation patterns mentioned in [53, 162]: (1) the *activity replacement* pattern, (2) the *activity reordering* pattern, and (3) the *activity repetition* pattern. Other high-level deviation patterns can be easily derived based on these patterns.

5.2.1 Activity Replacement Pattern

In practice, an execution of an activity may be replaced by an execution of another activity that is similar to the original one. Take for example the net in Figure 5.1. In a normal situation, the documents of a registered application are checked by an officer (check documents). However, if suspicious information is found in the application then a manager may decide to check the documents himself instead of the officer soon after both the fingerprints of the applicant are collected and the interview report of the applicant is obtained, i.e., by performing activity recheck. In this case, activity recheck *replaces* check document. Suppose that the trace for this case is $\sigma_2 = \langle \mathit{register}, \mathit{fingerprint}, \mathit{interview}, \mathit{recheck}, \mathit{decision} \rangle$. Figure 5.3 shows an optimal alignment between the trace and the net in Figure 5.1. The alignment shows that there are two low-level deviations in the trace. However, it does not show that activity check document is replaced with activity recheck.

We can check whether activity recheck is replaced with check document in trace σ_2 by computing an optimal alignment as explained in Chapter 4. First, we append transition ti_5 labeled recheck to the original net and copy both the input/output arcs of transition

register	*fingerprint*	*interview*	$\gg$	*recheck*	*decision*
register t_1	*fingerprint* t_2	*interview* t_3	*check document* t_5	$\gg$	*decision* t_6

Figure 5.3: An optimal alignment between trace $\sigma_2 = \langle register, fingerprint, interview, recheck, decision\rangle$ and the net of Figure 5.1 while using the standard likelihood cost function.

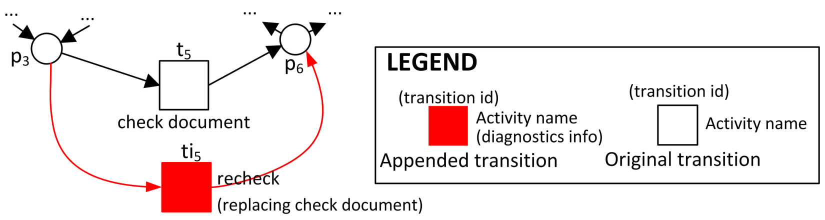


Figure 5.4: The result of appending ti_5 to the net shown in Figure 5.1. ti_5 models the replacement of activity check document with activity recheck.

Trace	*register*	*fingerprint*	*interview*	*recheck*	*decision*
Model	*register* t_1	*fingerprint* t_2	*interview* t_3	*recheck* ti_5	*decision* t_6
Diagnostics	–	–	–	check document *is replaced with* recheck	–

Figure 5.5: Top: An optimal alignment between trace $\sigma_2 = \langle register, fingerprint, interview, recheck, decision\rangle$ and the net of Figure 5.1 appended with the replacement pattern instance shown in Figure 5.6 while using the standard likelihood cost function, **Bottom:** Diagnostics obtained by translating the synchronous moves of appended transitions into high-level deviation diagnostics.

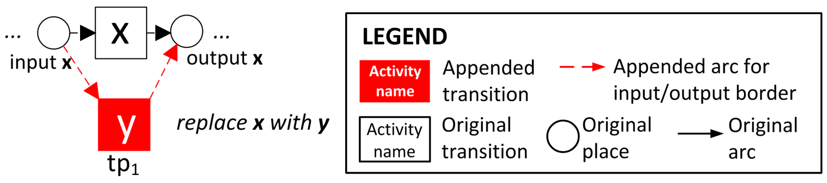


Figure 5.6: A deviation pattern for replacing an activity (x) with another activity (y).

labeled check document (t_5) as shown in Figure 5.4. This way, we explicitly model the replacement of activity check document with activity recheck. Second, we compute an optimal alignment between σ_2 and the appended net to check whether such a replacement occurs. An optimal alignment between them is shown in Figure 5.5. The fourth column of the optimal alignment in Figure 5.5 shows a synchronous move between the appended transition ti_5 with activity recheck. Such a synchronous move can be translated into high-level deviation diagnostics indicating that in σ_2 there is a replacement of activity recheck with activity check document.

The set of appended transitions and arcs in Figure 5.4 is an *instance* of a high-level deviation pattern that represents the replacement of an activity with another activity. Figure 5.6 shows the deviation pattern. The same pattern can be used to check the replacement of other pairs of activities. Transition tp_1 in the pattern has exactly the

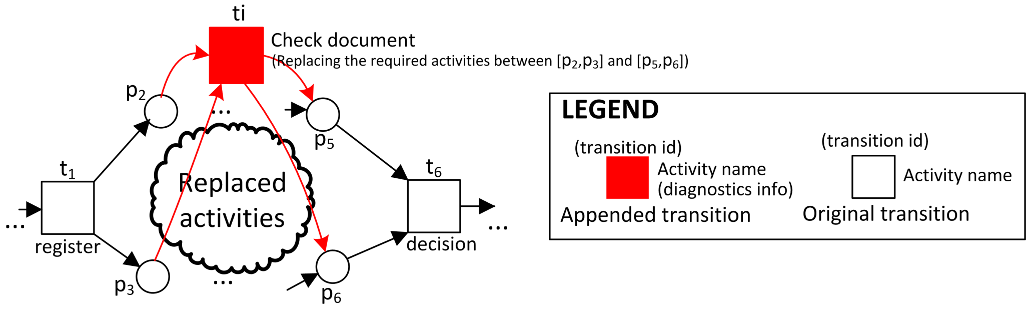


Figure 5.7: Modeling the execution of check document that replaces all activities in the fragment of the Petri net shown in Figure 5.1 between marking $[p_2, p_3]$ and $[p_5, p_6]$. Firing ti from marking $[p_2, p_3]$ leads to marking $[p_5, p_6]$

same set of input and output places as the transition in the fragment with label x, i.e., tp_1 is an input border as well as an output border of the pattern. The transition with label x is enabled if and only if transition tp_1 is enabled, and firing the transition labeled x from a marking m leads to the same marking as the one yielded by firing tp_1 from m.

The replacement pattern in Figure 5.6 can be further generalized to check the *replacements of sets of activities*. An example of an appended instance of this generalized pattern is shown in Figure 5.7. Figure 5.7 models an execution of activity check document that replaces all activities that should be performed between activity register and decision according to the original Petri net. Marking $[p_2, p_3]$ is reachable after transition t_1 (labeled register) is fired. The addition of transition ti to the original model allows a firing sequence $\langle t_1, ti \rangle$ from the initial marking of the original net that enables t_6 (labeled decision). Note that marking $[p_2, p_3]$ has to be reachable from the initial marking of the original Petri net and marking $[p_5, p_6]$ has to be reachable from marking $[p_2, p_3]$ to ensure that the set of reachable markings in the original net is preserved.

5.2.2 Activity Reordering Pattern

Process models show how activities are ordered. In practice, some activities can be reordered. For example, the order of activity fingerprint in a visa application handling process shown in Figure 5.1 may be swapped with interview if the fingerprint machine is not available right after the application is registered. A possible trace for this case is $\sigma_3 = \langle register, interview, fingerprint, check\ document, decision \rangle$. To check whether the two activities are reordered in the trace, we use the same approach explained in Section 5.2.1. We first append the original net with an instance of activity reordering pattern. Then, we compute an optimal alignment between the trace and the appended net.

Figure 5.8 shows a Petri net fragment that is appended to the original net in Figure 5.1. Firing transition ti_1 has the same effects as firing t_2 in the original net (ignoring place pi_1). Similarly, firing ti_2 has the same effect as firing t_3. Without the addition of place pi_1, the addition of ti_1 and ti_2 can be viewed as two instances of the activity replacement pattern. However, we can only say that activity fingerprint is swapped with interview if both replacements occur. The addition of place pi_1 ensures that transition ti_2 can only be fired if and only if transition ti_1 has been fired before, i.e., the place ensures ordering between appended transitions. Furthermore, firing ti_1 without firing ti_2 will leave a token in place pi_1. This implies that the final marking of the original net is not

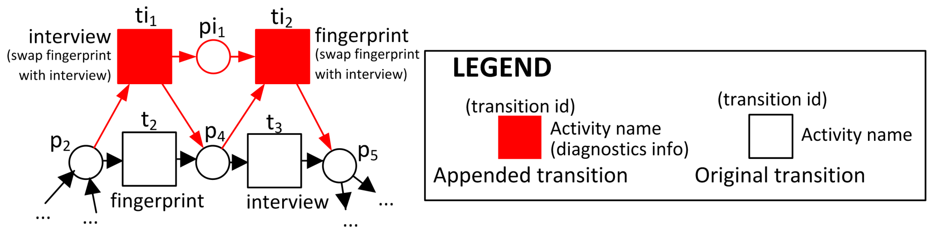


Figure 5.8: Modeling the swapping of activity fingerprint with interview in the net shown in Figure 5.1.

Trace	*register*	*interview*	*fingerprint*	*check document*	*decision*
Model	*register* t_1	*interview* ti_1	*fingerprint* ti_2	*check document* t_5	*decision* t_6
Diagnostics	–	fingerprint *is swapped with* interview		–	–

Figure 5.9: **Top:** An optimal alignment between trace $\sigma_3 = \langle register, interview, check\ document, fingerprint, decision \rangle$ and the appended net of Figure 5.8, **Bottom:** Diagnostics obtained from synchronous moves of appended transitions.

reachable. Thus, given a trace and a Petri net appended with an instance of the swapped activities (e.g., Figure 5.8), place pi_1 ensures that an alignment between them contains zero or more pairs of ti_1 and ti_2.

Figure 5.9 shows an optimal alignment between σ_3 and the appended net while using the standard likelihood cost function. The columns of the alignment that contain synchronous moves of the appended transitions (i.e., ti_1 or ti_2) show the locations of swapped activities. In this example, the pair of synchronous moves $(interview, ti_1)$ and $(fingerprint, ti_2)$ shows that activity fingerprint in the trace is swapped with activity interview.

Given a trace and a Petri net, the *activity reordering pattern* expresses activities in the trace that are not performed in the right order. Unlike the replacements of activities, the reordering of activities cannot be modeled using only a single transition (see e.g., Figure 5.8). Figure 5.10 shows an example of such a pattern, i.e., the reordering of two activities pattern. Given a trace and a Petri net, a pair (x, y) of activities in the trace are swapped if: (1) x occurs before y, and (2) swapping y with x in the trace yields a better fitting trace than the original trace with respect to the net. Swapping two activities x with y can be modeled as a pair of Petri net transitions with a strict ordering between them such that one of the transition must always occur before the other. It is easy to see that the appended transitions and place in Figure 5.8 is an instance of the pattern in

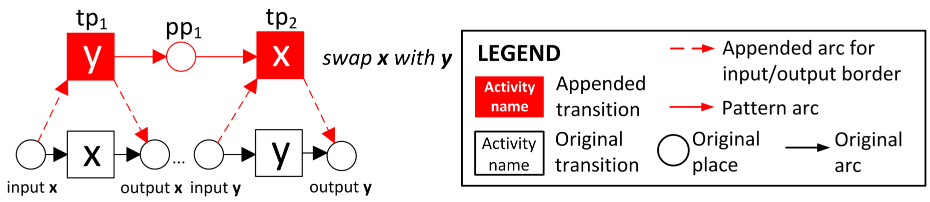


Figure 5.10: A deviation pattern for swapping an activity (x) with another activity (y).

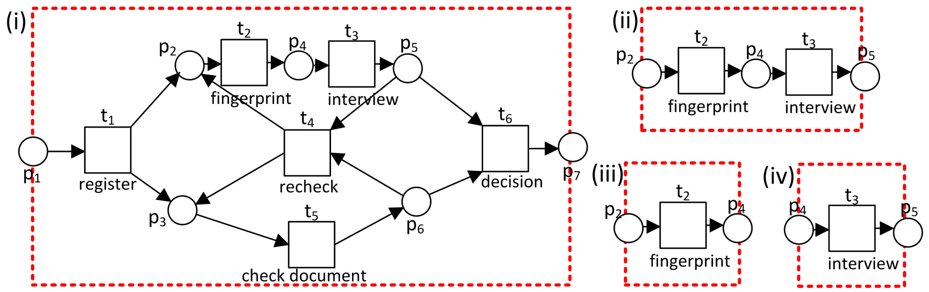


Figure 5.11: Four workflow fragments of the net shown in Figure 5.1. Only fragments (iii) and (iv) do not share any transition. Thus, swapping between the two fragments can be checked using the same pattern as Figure 5.10.

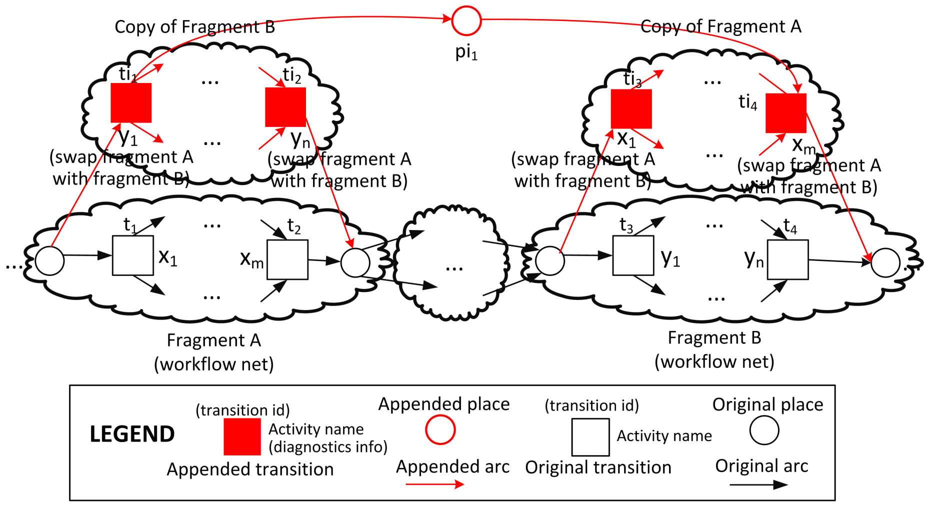


Figure 5.12: A Petri net, appended with transitions and places to identify swapping of activities in two sound workflow fragments of the net.

Figure 5.10.

The swapping pattern in Figure 5.10 can be further generalized to check whether two sound workflow fragments of a Petri net are swapped. A workflow fragment of a Petri net is a workflow subset of the net with a unique start place and end place where all transitions, places, and arcs between the start and end place are preserved. Figure 5.11 shows some examples of workflow fragments of the net shown in Figure 5.1. Note that the complete net is also a workflow fragment. We can check whether any pair of the fragments that share no transition is swapped using the similar pattern shown in Figure 5.10. In Figure 5.11, the only pair of fragments that share no transition is the pair of fragments (iii) and (iv).

Figure 5.12 shows an example of a Petri net, appended with transitions and places to identify swapped activities in two sound workflow fragments of the net. In the figure, the swapped activities in two fragments A and B are modeled by copying each fragment and place them in a reversed order. Place pi_1 is added between the fragment copies to connect the start transition of the first fragment copy to the final transition of the other fragment. Sound workflow fragment is a sufficient requirement to guarantee that

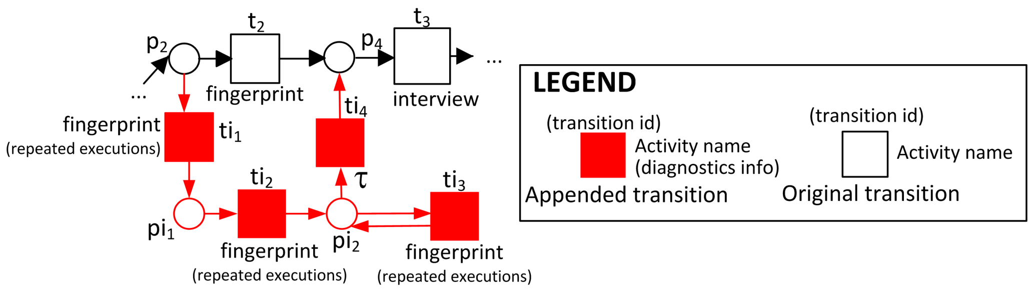


Figure 5.13: Appended transitions and places to check the repetition of activity fingerprint in the process shown in Figure 5.1.

Trace	*register*	*fingerprint*	*fingerprint*	*fingerprint*	$\gg$	*interview*	*check document*	*decision*
	register	*fingerprint*	*fingerprint*	*fingerprint*		*interview*	*check document*	*decision*
Model	t_1	ti_1	ti_2	ti_3	ti_4	t_3	t_5	t_6
Diagnostics	–	*repetitive execution of* fingerprint				–	–	–

Figure 5.14: Top: An optimal alignment between trace $\sigma_4 = \langle register, fingerprint, fingerprint, fingerprint, interview, check\ document, decision\rangle$ and the appended net shown in Figure 5.13, **Bottom:** Diagnostics obtained from movements of appended transitions.

the movements on the copied fragments can be translated back to high-level deviations, although it is not a necessary condition. The requirement guarantees that if a transition in one copied fragment is fired according to an alignment then both the fragment and its pair must be "completed", i.e., either a complete firing sequence of both of the copied fragments are observed or not at all.

5.2.3 Activity Repetition Pattern

In case of emergency, an activity that must be executed once according to a process model may be performed multiple times. Take for example the visa application process in Figure 5.1. Suppose that the fingerprints of a visa applicant needs to be scanned multiple times because of a technical problem on the fingerprint scanner. Trace $\sigma_4 = \langle register, fingerprint, fingerprint, fingerprint, interview, check\ document, decision\rangle$ is the trace of the case. It is easy to see that activity fingerprint is performed multiple times in the trace. We check the multiple occurrences of activity fingerprint in σ_4 by first appending the original net with new places and transitions as shown in Figure 5.13.

Transition ti_1 in Figure 5.13 represents the start of multiple execution of activity fingerprint. Transition ti_2 is added after ti_1 such that any alignment that contains ti_1 must also contain ti_2, i.e., a repeating fingerprint activity contains of at least two executions. Transition ti_3 is added to explicitly represent the third and consecutive occurrences of activity fingerprint, and invisible transition ti_4 is an output border that marks the end of the repetition.

Figure 5.14 shows an optimal alignment between the trace and the appended net shown in Figure 5.13. The occurrence of a synchronous move $(fingerprint, ti_1)$ in the optimal alignment of Figure 5.14 indicates the start of a repetitive execution of fingerprint while it is not allowed according to the original net. The move on model $(\gg, ti_4)$

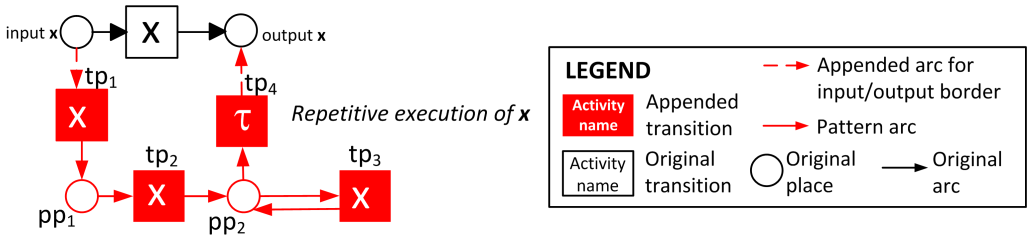


Figure 5.15: A deviation pattern for repetition of activity (x).

in the alignment explicitly indicates the end of the repetitive execution. Other synchronous moves of the added transitions are the repetitive execution of fingerprint (i.e., synchronous moves (*fingerprint*, ti_2) and (*fingerprint*, ti_3)).

The net in Figure 5.15 is an example of *activity repetition patterns*. The pattern models a repeated occurrences of an activity that should only occur once. Figure 5.13 is an instance of the pattern. A unique characteristic of this type of deviation is that the number of repetitions is unrestricted. Therefore, the repetition pattern contains a loop construction. Similar to the activity reordering pattern, this pattern can also be extended to check the repetitions of a sound workflow net by replacing all colored transitions in the pattern with the fragment.

5.2.4 Constructing Patterns

To this point, we described three deviation patterns and their extensions. It is obvious that there are many other patterns that one can make to check high-level deviations. Nevertheless, we provide some guidelines on how to construct new deviation patterns based on the three patterns explained before. Given a trace, a Petri net, and a deviation pattern, any deviation modeled by an appended instance of the pattern needs to be translated from an alignment between the trace and the net to higher-level diagnostics. To ensure that this is possible, we recommend deviation patterns with a unique *initial transition* and *final transition*. All complete firing sequences of a deviation pattern with a unique initial/final transition start with the initial transition, end with the final transition, and contain no initial/final transition in between. This way, a repetitive activity deviation is "marked" between a synchronous move of the initial transition and a synchronous move of the final transition.

All three patterns that we introduced in this section were created by following these guidelines. For example, tp_1 in Figure 5.6 is both the initial and the final transition of the activity replacement. The pattern only contains one transition, therefore it is easy to see that all firing sequences of the pattern starts and ends with tp_1. The initial and final transition of the activity repetition pattern shown in Figure 5.15 are two different transitions. The initial transition of the pattern is transition tp_1 and the final transition of the pattern is the invisible transition tp_4. The pattern is a sequence of transitions, thus it is trivial to see that all complete firing sequences of the pattern start with tp_1 and end with tp_4. For all Petri nets appended with an instance of the pattern, a complete firing sequence of the net that contains the instance of tp_1 also contains the instance of tp_4 (e.g., transition ti_1 and ti_4 in Figure 5.13).

5.3 Identifying the Root Causes of Deviations

Given a trace and a Petri net, Section 5.2 describes several high-level deviation patterns and how they can be used in order to *check occurrences of high-level deviations* in the trace. In practice, this is rarely the case. Instead, we would like to identify whether some high-level deviations occur in the trace without any prior knowledge.

Suppose that there are multiple high-level deviations of interest and all deviation patterns follow the guidelines mentioned in Section 5.2.4. We first append all high-level deviation pattern instances to the net and then compute an optimal alignment between the appended net with the original trace. To construct the alignment, we use a new cost function of movements based on a predefined cost of low-level deviations (moves on log, moves on model) and costs of high-level deviations. Let PA be a high-level deviation pattern and let PA' be an instance of PA, appended to a Petri net. The cost of the deviation represented by PA is assigned to the cost of the synchronous move of the initial transition of PA'. The cost of synchronous moves for all other transitions in PA' are set to 0. Furthermore, we assign a very high cost ($+\infty$) to all moves on model of all non-invisible transitions in PA'. This way, for all appended Petri net, optimal alignments between the trace and the net will not contain any movement for which only some transitions of pattern instances appear as synchronous moves, i.e., either all or no transitions of the instance are included.

Take for example trace $\sigma_5 = \langle$*register*, *interview*, *fingerprint*, *fingerprint*, *fingerprint*, *fingerprint*, *recheck*, *decision*$\rangle$ and the net shown in Figure 5.1. Suppose that there are three high-level deviations of interest: (1) swapping fingerprint with interview, (2) replacing check document with recheck, and (3) repetitive execution of fingerprint. All high-level deviations have cost 1 and all other deviations follow the standard cost function. Figure 5.16 shows the net after appended with all three instances of the pattern of interest. The cost of synchronous moves are annotated in the figure. Figure 5.17 shows an optimal alignment between the trace and the appended net using the derived cost function.

We identify all high-level deviations from the optimal alignment in Figure 5.17 by looking at all synchronous moves with the initial transition of some pattern instances. There are two columns in the alignment that contain such initial transitions. Transition ti_1 in column 3 is an instance of the initial transition of the repetition pattern (see Figure 5.15), and transition ti_5 in column 9 is an instance of the initial transition of the replacement pattern (see Figure 5.6). Thus, there are two high-level deviations that occur in σ_5: (1) A repetition of fingerprint, and (2) A replacement of check document with recheck. Furthermore, there are also two low-level deviations: (1) a move on log of activity interview and (2) a move on model of transition t_3. The total cost of deviation in the alignment is 4. Notice that one can also consider that activity fingerprint is swapped with interview, but more low-level deviations will be identified as shown in Figure 5.18 and it yields higher total cost of deviations (i.e., 5 deviations). This example shows that the approach is able to identify root causes of deviations.

The assignment of $+\infty$ to all moves on model of all non-invisible pattern instance transitions is crucial. Take for example a trace $\sigma_6 = \langle register, fingerprint, interview, decision\rangle$ of the net in Figure 5.1. In the trace, activity check document is skipped while it should be performed according to the model. Since the deviation is unknown in advance, we are looking for the same set of all interesting high-level deviations as the one described in the previous example. Thus, we compute an optimal alignment between σ_6 and the

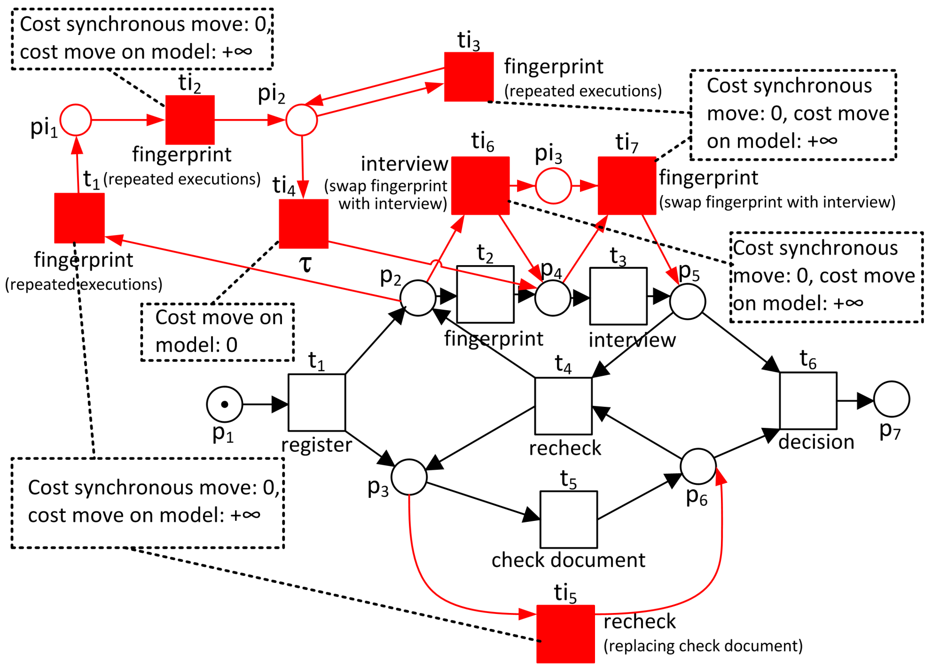

Cost synchronous move: 0, cost move on model: +∞
ti3
fingerprint (repeated executions)
ti2
pi2
pi1
Cost synchronous move: 0, cost move on model: +∞
fingerprint
t1 (repeated executions)
ti4
interview (swap fingerprint with interview)
ti6
pi3
ti7
fingerprint (swap fingerprint with interview)
fingerprint (repeated executions)
τ
p2
t2
p4
t3
p5
Cost synchronous move: 0, cost move on model: +∞
Cost move on model: 0
fingerprint
interview
t1
t4
t6
p1
register
recheck
decision
p7
Cost synchronous move: 0, cost move on model: +∞
p3
t5
p6
check document
ti5
recheck (replacing check document)

Figure 5.16: The net in Figure 5.1, appended with pattern instances in Figure 5.6, Figure 5.10, and Figure 5.15. The costs of moves on model of all colored non-invisible transitions, i.e., all added visible transitions, are $+\infty$.

Trace	*reg*	*int*	*fp*	*fp*	*fp*	*fp*	$\gg$	$\gg$	*reck*	*dec*
Model	*reg* t_1	$\gg$	*fp* ti_1	*fp* ti_2	*fp* ti_3	*fp* ti_3	ti_4	*int* t_3	*reck* ti_5	*dec* t_6
Diagnostics	–	–	*repetitive execution of* fingerprint					–	check document *is replaced with* recheck	–

LEGEND
reg : *register* *fp* : *fingerprint* *int* : *interview* *dec* : *decision* *reck* : *recheck*

Figure 5.17: Top three rows: An optimal alignment between trace $\sigma_5 = \langle register, interview, fingerprint, fingerprint, fingerprint, fingerprint, recheck, decision\rangle$ and the appended model in Figure 5.16. **Bottom two rows:** high-level deviation diagnostics based on synchronous moves of appended extra transitions.

appended net shown in Figure 5.16. However, instead of assigning cost $+\infty$ to all moves on model of appended non-invisible transitions, we assign cost of 1 to all of them. Figure 5.19 shows a possible optimal alignment between σ_6 and the net shown in Figure 5.16. As shown in the alignment, column 4 cannot be translated back to high-level deviations of replacing check document with recheck, because no activity occurs in column 4 (i.e., $\pi_1((\gg, ti_5)) = \gg$).

This example shows that not all optimal alignments between a trace and an appended Petri net can be translated to high-level deviations. However, the assignment of cost $+\infty$ to all moves on model of appended non-invisible transitions ensures that they cannot be aligned unless there is a corresponding event in the log. This way, for all appended pattern instances, optimal alignments between the trace and the appended Petri net will not contain any movement for which only some transitions of the patterns appear as synchronous moves, i.e., either all or no transitions of the instance are included.

Recall the previous example. Suppose that we compute an optimal alignment be-

Trace	*reg*	*int*	*fp*	*fp*	*fp*	*fp*	*reck*	*dec*
Model	*reg* t_1	*int* ti_6	*fp* ti_7	$\gg$	$\gg$	$\gg$	*reck* ti_5	*dec* t_6
Diagnostics	–	fingerprint *is swapped with* interview		–	–	–	check document *is replaced with* recheck	–

LEGEND
reg : *register* *fp* : *fingerprint* *int* : *interview* *dec* : *decision* *reck* : *recheck*

Figure 5.18: Top three rows: A non-optimal alignment between trace $\sigma_5 = \langle$*register*, *interview*, *fingerprint*, *fingerprint*, *fingerprint*, *fingerprint*, *recheck*, *decision*$\rangle$ and the appended model in Figure 5.16, showing activity fingerprint is swapped with interview. **Bottom two rows:** high-level deviation diagnostics based on synchronous moves of appended extra transitions.

register	*fingerprint*	*interview*	$\gg$	*decision*
register t_1	*fingerprint* t_2	*interview* t_3	*recheck* ti_5	*decision* t_6

Figure 5.19: A possible optimal alignment between trace $\sigma_6 = \langle \mathit{register}, \mathit{fingerprint}, \mathit{interview}, \mathit{decision} \rangle$ and the appended Petri net shown in Figure 5.16 with respect to the standard cost function.

tween σ_6 and the net in Figure 5.16 using the modified standard cost function. Figure 5.20 shows an optimal alignment between σ_6 and the net of Figure 5.17 with respect to the modified cost function. As shown by the figure, the optimal alignment correctly identifies a low-level deviation: check document (t_5) is skipped in σ_6.

The decision on which pattern instances need to be appended to a Petri net is taken by a process expert. This way, we ensure that the obtained high-level deviation diagnostics are meaningful from the expert point of view. Allowing arbitrary pattern instances may yield diagnostics that do not make any sense. For example in the Petri net of Figure 5.1, replacing activity check document with recheck make sense because both activities have a similar definition. However, replacing activity check document with interview does not really make sense as both activities are semantically different. Although the expert need to determine a set of high-level deviations of interest, the translation from the set to pattern instances can be automated. This way, the technique described in this chapter can be used by a process expert without necessarily knowing about high-level deviation patterns and the mechanics behind them.

5.4 Related Work

Checking the compliance of a set of rules to an observed execution has some similarities with identifying high-level deviations from a given trace and a given Petri net. A Petri net can be viewed as a set of low-level compliance rules. Each rule consists of a reachable marking of the net and the set of activities that must not occur directly from the marking.

register	*fingerprint*	*interview*	$\gg$	*decision*
register t_1	*fingerprint* t_2	*interview* t_3	*check document* t_5	*decision* t_6

Figure 5.20: The only optimal alignment between trace $\sigma_6 = \langle \mathit{register}, \mathit{interview}, \mathit{check\ document}, \mathit{decision} \rangle$ and the appended Petri net shown in Figure 5.16 with respect to a modified standard cost function where the cost of $(\gg, ti_6), (\gg, ti_7)$ are $+\infty$.

If the trace does not fully comply with the set of rules imposed by the net, there is either an activity that occurs in the trace while it must not occur according to the net or the other way around. A set of high-level deviations is a set of rules defined on top of the set of rules already imposed by the net. If the trace violates a high-level compliance rule, there are some deviations with respect to low-level compliance rules but not necessarily the other way around, as shown in the example given in Section 5.1.

Many approaches to diagnose high-level deviations between observed and modeled behavior have been proposed in the area of *compliance checking*. In particular, we refer to the approaches on backward compliance checking [90]. Given a set of predefined constraints and rules and a set of recorded executions, backward compliance checking techniques verify if the executions are in accordance with the set of constraints/rules. In [17], Blaze et al. propose a conceptual approach to perform a compliance checking for Trust Management Systems. The approach underlines the importance of having not only a compliance checker that checks the compliance between a set of policies (i.e., rules) and a set of actions executed so far (i.e., observed behavior), but also provides proofs of compliance that can be explained, formalized, and proven correct.

In [33], Chesani et al. propose an approach based on computational logic to check whether a careflow process execution conforms to a set of predefined rules. The rules are formalized in form of computational logics and are checked against process execution using the $\mathcal{S}CIFF$ framework [32]. If some of the rules are not satisfied, the framework points out the rules that are not satisfied. Giblin et al. also propose a similar compliance checking approach based on real-time temporal object logics [66]. Given a set of rules in form of Timed Propositional Temporal Logics [15], the approach provides a yes/no answer to the question whether the rules are followed by process executions. Furthermore, the approach also proposes a concept model called REALM to describe the relationships between entities in the environment where the set of rules is applied. This way, the rules may also contain relationship between entities.

Governatori et al. proposed an approach to check the compliance of business process executions to a set of business process contracts [70]. The contracts are formalized using the logic based formalism named FCL [69]. Other than providing a subset of fully compliant contracts and other subset of non-compliant contracts from the set of contracts, this work also introduces the concepts of ideal, sub-ideal, non-ideal, and irrelevant situations to measure the degree of compliance between the contracts and executions according to the set of contracts that have been violated and the ability of the business process to fix occurred violations.

In the area of process mining, some approaches related to compliance checking that can also be used to diagnose high-level deviations exist. Aalst et al. proposed an LTL-based approach to check the compliance of recorded process executions in form of event logs to a set of predefined rules [178]. Montali et al. propose an approach to check the compliance between a predefined set of declarative rules to recorded process executions [112]. In [191], Werf et al. propose an approach to check the compliance of a process execution to a predefined set of rules that takes into account process context using ontology.

All of the previously mentioned approaches only provide yes/no answer to compliance rules without providing further diagnostics information. Ramezani et al. introduces a compliance checking technique with precise diagnostics information [137]. In [137], an exhaustive list of control-flow compliance rules is provided in Petri net formalism. Given a trace and a rule in Petri net formalism, the approach not only provides a yes/no answer whether the trace complies to the rule, but also detailed diagnostics information on where the deviation occurred. Note that if more than one compliance rule is checked against a trace, diagnostics are shown for each rule independent of the others. This approach is extended to consider data in [138]. Similarly, Banescu et al. [16] introduce an approach to measure compliance between a given process model and its executions based on the well-known Levenshtein distance [98]. Other than compliance checking, the approach also provides diagnostics if deviations exist in the executions. However, the approach needs to list all activity sequences allowed by the model, which is not feasible if the model contains loop and therefore allows for infinitely many sequences.

While the approaches of Ramezani [137] and Banescu [16] require *imperative* process models, de Leoni et al. [43] introduces an approach to check conformance for *declarative* models. Given an event log and a Declare model [124], de Leoni et al. proposes an approach to check conformance between them [43]. The result of conformance checking does not only shows explicitly where the deviations are, but also the possible root causes of deviations. Similar work is also proposed by Maggi et al. [101] for monitoring declarative constraints in runtime settings.

From all approaches mentioned in this section, given a trace that does not comply with a set of compliance rules, there is not yet an approach that provides diagnostics on which deviations are the root cause of non compliances (if they exist). In Section 5.3, we describe an approach that does not only provide diagnostics on high-level deviations, but also identifies the root causes of deviations in cases where more than one high-level deviations occur.

5.5 Experiments

In this section, we show some experimental results used to validate and test the proposed approach. The approach is implemented in ProM 6 [198]. We performed two sets of experiments. The goal of the first set of experiments is to show that the approach can be used to identify high-level deviations. The second set of experiments use logs and models taken from real-life cases to show the applicability of the approach. The results of the first set of experiments are described in Section 5.5.1, and the results of the second set of experiments are shown in Section 5.5.2.

5.5.1 Artificial Logs and Models

For this set of experiments, we use the artificial process model shown in Figure 5.1. 30 perfectly fitting event logs were generated from the model, each log consists of 100 traces with 5 to 15 activities per trace. Then, we created 11 variants of non fitting logs from the perfectly fitting logs by removing, inserting, swapping, and replacing some activities with other activities randomly. Each variant has different number of swapped (swap), replaced (rep), inserted (ml), and removed activities (mm). For each variant, 30 event logs were constructed. Then, we computed optimal alignments for all traces in all logs and measure the average total cost per trace and the computation time required per event log.

In this set of experiments, three different possible scenarios to diagnose deviations were compared. In the first scenario, we consider that all swapping and replacement of all activities are possible. In the second approach, we define a subset of high-level deviations that is considered to be meaningful, thus the set of high-level deviations (i.e., swapped and replaced activities) of interest is a subset of the high-level deviations in

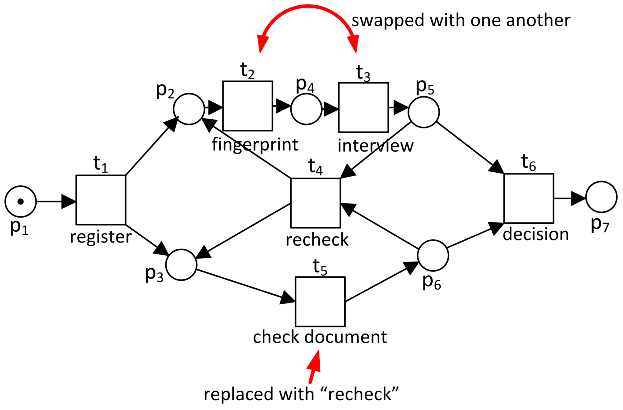


Figure 5.21: Selected high-level deviations that provide meaningful diagnostics: swapping interview with fingerprint (and vice versa), and replacing check document with recheck.

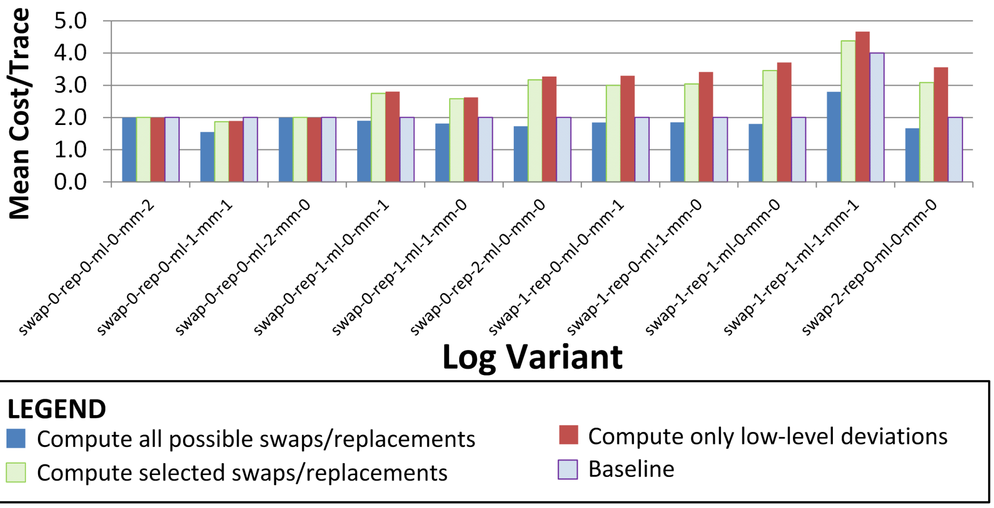


Figure 5.22: Mean average cost per trace. Each bar is computed from 30 experiments using randomly generated noisy logs. Baseline shows expected mean cost if the noise are known in advance.

the first scenario. Figure 5.21 shows the model used in this set of experiments and the selected high-level deviations. As shown in the figure, only the swapping of activity interview with fingerprint (and vice versa) and the replacement of check document with recheck were investigated in the second approach. In the third approach, only low-level deviations were computed (i.e., moves on log, moves on model). Note that the first two approaches also consider low-level deviations. In all experiments, we use a cost function where all low-level deviations have cost of 1 (the same as the standard cost function) and all high-level deviations have cost of 1, e.g., replacing activity a with b has cost 1 and swapping activity a with b has cost 1 for all activities a and b in the logs.

Figure 5.22 and Figure 5.23 show the results of the experiments. The baseline value for a variant of experiment is taken from the expected cost if the operation to make the variant non-fitting was known in advance, e.g., the baseline value for variant "swap-1-rep-1-ml-0-mm-0" is 2 because 1 random swap and 1 replacement were performed to all

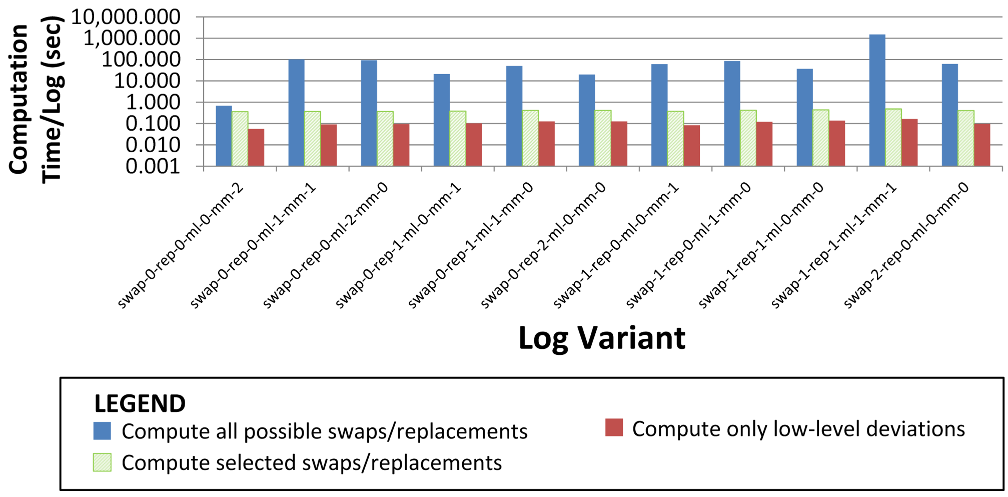


Figure 5.23: Mean computation time per log. Each bar is computed from 30 experiments using randomly generated noisy logs. The y-axis is shown in logarithmic scale.

logs created using this variant. Figure 5.22 shows that the mean average cost per trace of the approach that considers all possible swaps/replacements is always the lowest, followed by the second scenario that only consider some meaningful high-level deviations. The approach where only low-level deviations are identified always yields the highest mean cost. In many variants (e.g., "swap-0-rep-0-ml-1-mm-0" and "swap-1-rep-1-ml-1-mm-1") the first approach provides a lower mean cost than the baseline because some combinations of low-level-deviations are identified as high-level-deviations. In contrast, in most cases the approach that computes only low-level deviations provides a higher mean cost than the baseline. An exception occurs in variant "swap-0-rep-0-ml-1-mm-1", because in some traces the removed activity is the same as the appended extra activity.

Figure 5.23 shows the average time spent to identify all deviations (i.e., to compute optimal alignments) in event logs for all variants. We clearly see that the time required by the first scenario (i.e., the one that check for all possible swaps/replacements) is much higher than the other approaches. In all variants, the computation time for the first scenario is more than 50 times higher than the time required by the approach that only computes low-level deviations. In some variants, it took around 100,000 seconds (±27.7 minutes) on average to diagnose all deviations in an event log that only contains 100 traces. The scenario that considers only some subsets of all possible swaps/replacements requires slightly more computation time compared to the one that only computes low-level deviations. However, the average computation time of the scenario is under a second in all variants. These results suggest that the selection of high-level deviations before applying the proposed approach is crucial.

In the Section 5.5.2, we use the proposed approach to identify high-level deviations in a real-life case, taken from a Dutch municipality.

5.5.2 Real-life Case

The experimental results in Section 5.5.1 show that the proposed approach is potentially computationally expensive. Nevertheless, in this section we show that the approach is applicable in practice. We took a pair of a log and a model from a Dutch municipality

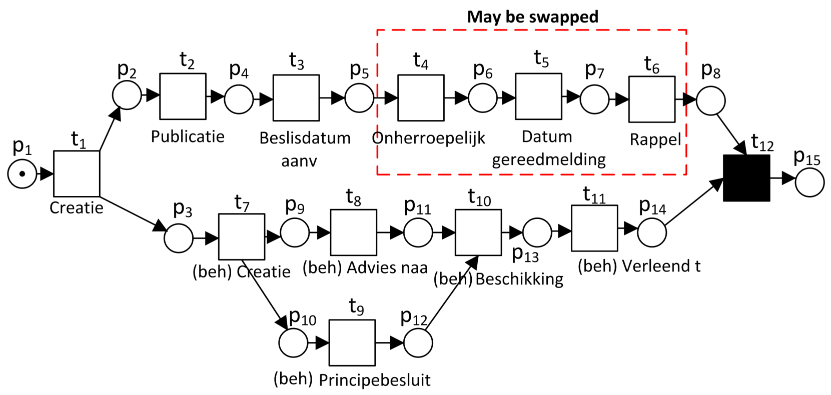


Figure 5.24: Real-life process model used in the experiment. The experiment was performed to identify cases where “Onherroeplijk”, “Datum gereedmelding”, and “Rappel” are swapped.

where some high-level deviations may have occurred. The log and model refers to a building permit handling process. The model is shown in Figure 5.24.

The process starts when a municipality officer creates (Creatie) a building permit application. The officer needs to publish (Publicatie) the application to notify people especially the ones that live near the location of the planned building, in case they object to the application. In parallel, a backend process occurs (i.e., the bottom branch of the process). The officer creates an internal request ((beh) Creatie) and seeks for advice (written/direct) from some experts about the application ((beh) Advies naa). In parallel, he can make a temporary decision on how to proceed with the original application. After a while, a decision is made ((act) Beschikking) and an acceptance document is made ((beh) Verleend t). In parallel, the date of decision is published (Beslisdatum anv) and after a while a definite decision is made (Onherroepelijk). Typically, the definite decision is followed by the acceptance statement that the building is finished (Datum gereedmelding) and withdrawal of the application (Rappel). However, for exceptional cases the acceptance statement and withdrawal may be performed in a different order. In this experiment, we investigate how often such reordering occurs.

The log contains 3,181 traces and we used the approach proposed in this chapter to investigate possible swappings between activities Onherroeplijk, Datum gereedmelding, and Rappel. We identified 28 traces where high-level deviations of swapping between the three activities occurred. A screenshot of the deviation diagnostics taken from one of the traces is shown in the top of Figure 5.25. As shown in the figure, instead of identifying only low-level deviations, the approach manages to also identify high-level deviations between activities Onherroepelijk and Datum gereedmelding. This particular insight is interesting for process experts, because it implies that there is a building permit application for which the building is finished before a definite decision about the application is issued. Compare this diagnostics with the diagnostics provided without identifying high-level deviations in the bottom of Figure 5.25. If we only consider low-level deviations, we get diagnostic information that an activity Onherroepelijk is not performed at the moment it should (as it is identified as a move on log), but we cannot easily draw a conclusion that it is actually swapped with Datum gereedmelding without manual analysis. Figure 5.26 shows the diagnostics obtained from the top figure of Figure 5.25.

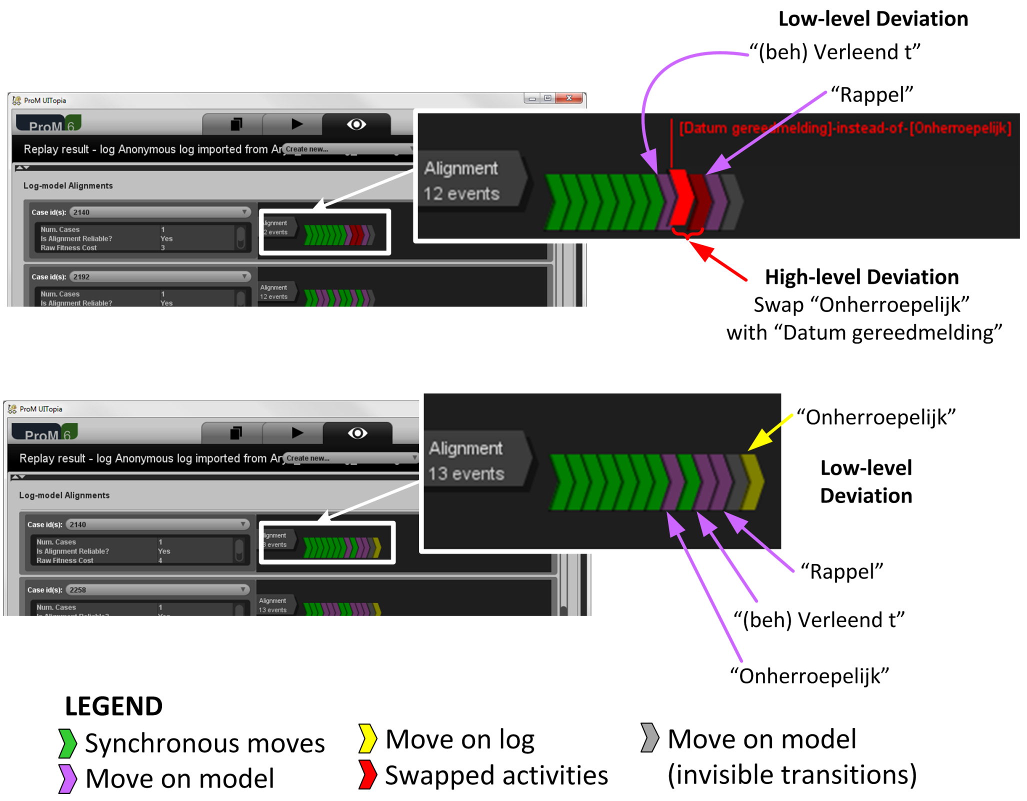


Figure 5.25: Screenshot of the implemented approach in ProM, showing a comparison of diagnostics provided by considering high-level deviations (top) and without (bottom) in the experiment.

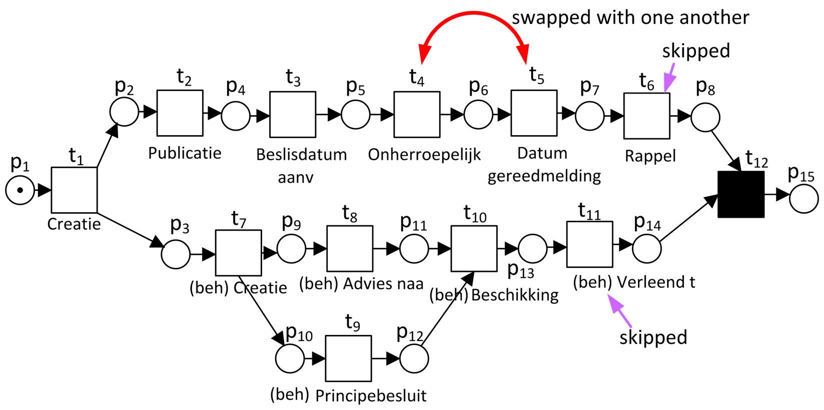


Figure 5.26: Projection onto process model of the diagnostics shown in Figure 5.25 (considering high-level deviations).

5.6 Conclusion

Experience shows that comparing process models and their executions in a non-strict environment can reveal interesting information about deviations that can be used for further analysis. On the one hand, information systems often record process executions at a rather low-level of detail, while on the other hand management is mostly interested in high-level diagnostics. Thus, there is a gap between analysis results that often rely on low-level information and management who utilizes the information.

In this chapter, we described an extension of alignments to check high-level deviations, hence bridging the gap between the abundance of low-level data and the need for high-level diagnostics. We showed that many high-level deviations can be described in the form of a combination of low-level deviations, i.e., deviation patterns. Furthermore, we showed how alignments can be exploited to identify root causes of deviations, taking into account multiple high-level deviations as well as low-level deviations all at once. Although the approach is computationally expensive, experiments showed that the approach can be used to handle logs and models with real-life complexity by assuming domain knowledge.

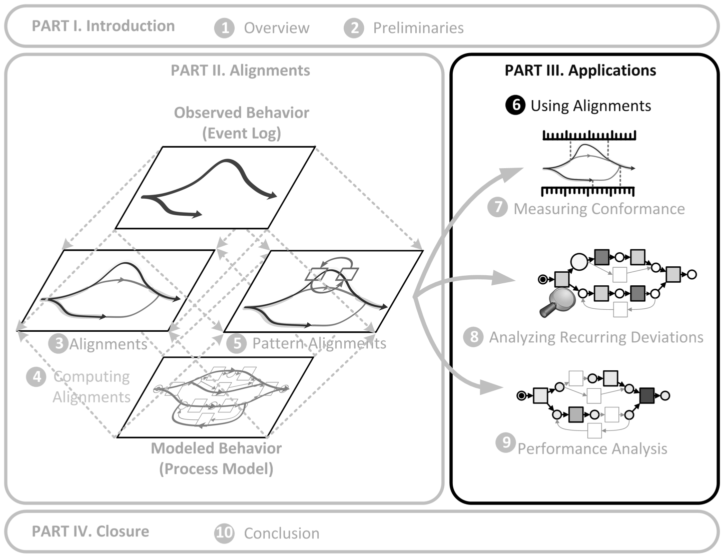
We emphasize that the selection of high-level deviations is very critical in this approach. A proper selection of high-level deviations does not only guarantee that the obtained diagnostics are meaningful, but is also relevant from a computational point of view.

Part III

Applications

Chapter 6

Using Alignments



6.1 Introduction

Given a trace and a Petri net, the concept of alignments offers a robust and flexible way to relate the observed behavior in the trace to the behavior modeled in the net. An alignment between a trace and Petri net is robust to peculiarities of process models such as invisible/duplicate transitions and complex control-flow patterns because activities in the trace are explicitly mapped to transitions allowed by the net. This explicit mapping provides more than just explicit identification of deviations, as we can also view the trace as a "path" through the process model by ignoring the moves on log. Such a model-based perspective of traces enables many types of analysis based on both traces and their corresponding Petri nets.

In this chapter, we highlight the role of alignments in various types of analysis based on both observed behavior and modeled behavior. In Section 6.2, we show a typical

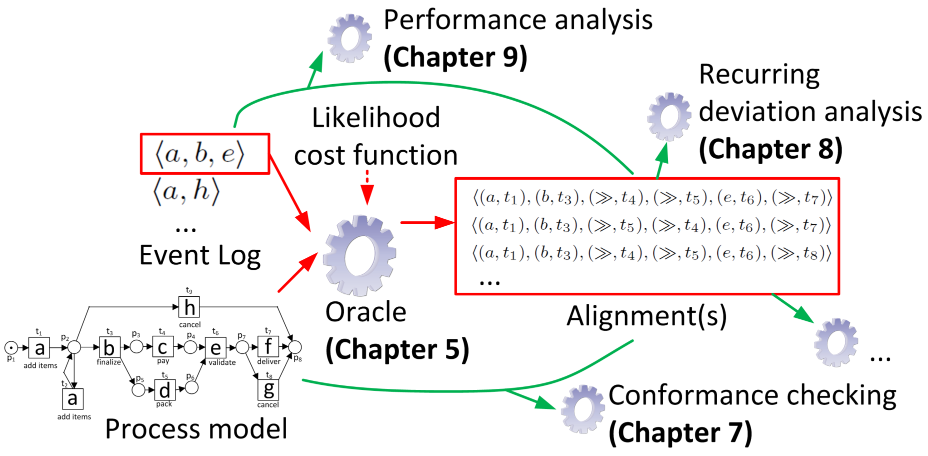


Figure 6.1: General approach for using alignments in various types of analysis.

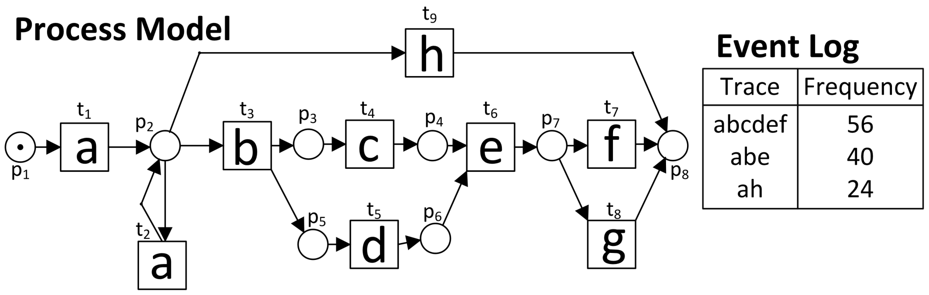
scenario using alignments. In Section 6.3, we define a metric to measure the quality of alignments between a given event log and a given Petri net. This metric serves as a measurement of quality of all later analyses based on alignments (ranging from conformance to performance analyses). In Section 6.4, we show possible extensions to take into account other information in event logs than just activity names.

6.2 General Use of Alignments

Many approaches in literature clearly separate model-based analysis and data-related analysis. For example, approaches to verify safety properties of processes (e.g., deadlock, livelock) [36, 76] and simulation [186], are based on process models without directly considering real data obtained from executions in reality. In contrast, data-driven approaches like data mining techniques [78] often discard process models. Given a trace and a Petri net, an alignment between them provides a way to relate activities in the trace to transitions of the net. Such a relation can be further exploited to enable both model-based analysis that take data into account and process-aware data analysis.

Figure 6.1 shows a general approach for using alignments. Given an event log and a process model, an oracle function maps each trace in the log to a set of alignments relating traces to paths in the model. An oracle function may use a likelihood cost function to assign probabilities of alignments. However, as discussed in the end of Section 3.4, this is not mandatory. The oracle function is used to construct a set of alignments for each trace in the log and the model. This set of alignments is used for various types of analysis. In Chapter 7, we show how alignments can be used to measure the conformance between an event log and a process model. The proposed approaches in Chapter 7 mainly use the alignments between each trace in the log with the model and the original process model. Most techniques to analyze the recurring deviations in Chapter 8 exploit alignments without considering either event logs or process models. The performance analysis approach in Chapter 9 requires the timestamps of events. Thus, both alignments and event logs are required to perform performance analysis.

In Chapter 4, we explained some approaches to compute optimal alignments effi-



Trace	Frequency
abcdef	56
abe	40
ah	24

Figure 6.2: Left: The model of an online transaction in an electronic bookstore, shown previously in Figure 4.10 with relabeled activities. **Right:** an event log of the model.

Table 6.1: All optimal alignments between all traces and net N in Figure 6.2 using the standard likelihood cost function.

σ	$\Gamma^o_{\sigma,N}$	**Label**
$\langle a,b,c,d,e,f\rangle$	$\langle(t_1,a),(t_3,b),(t_4,c),(t_5,d),(t_6,e),(t_7,f)\rangle$	γ_1
$\langle a,b,e\rangle$	$\langle(a,t_1),(b,t_3),(\gg,t_4),(\gg,t_5),(e,t_6),(\gg,t_7)\rangle$	γ_2
	$\langle(a,t_1),(b,t_3),(\gg,t_5),(\gg,t_4),(e,t_6),(\gg,t_7)\rangle$	γ_3
	$\langle(a,t_1),(b,t_3),(\gg,t_4),(\gg,t_5),(e,t_6),(\gg,t_8)\rangle$	γ_4
	$\langle(a,t_1),(b,t_3),(\gg,t_5),(\gg,t_4),(e,t_6),(\gg,t_8)\rangle$	γ_5
	$\langle(a,t_1),(\gg,t_9),(b,\gg),(e,\gg)\rangle$	γ_6
	$\langle(a,t_1),(b,\gg),(e,\gg),(\gg,t_9)\rangle$	γ_7
	$\langle(a,t_1),(b,\gg),(\gg,t_9),(e,\gg)\rangle$	γ_8
$\langle a,h\rangle$	$\langle(a,t_1),(h,t_9)\rangle$	γ_9

ciently. These approaches can be used to construct oracle functions. Take for example the event log and Petri net N in Figure 6.2. Table 6.1 shows the set of all optimal alignments between all traces in the log of Figure 6.2 and the net using the standard likelihood cost function. For the sake of readability, we use the alignment labels as shown in the table to refer to particular alignments in the remainder of this chapter.

Using the approaches in Chapter 4, there are at least three oracle functions that can be constructed for the log and net in Figure 6.2. Given a trace, a Petri net, and a likelihood cost function, the approach explained in Section 4.4 computes one optimal alignment between the trace and the net. We use this approach to construct an oracle function that assigns *100% probability for one of the optimal alignments* between each trace in the log and net N. Suppose that the optimal alignments between the following traces: $\langle a,b,c,d,e,f\rangle,\langle a,b,e\rangle,\langle a,h\rangle$, and net N are γ_1,γ_2, and γ_9 respectively. orc^1 is an oracle function constructed from the alignments such that $orc^1(\langle a,b,c,d,e,f\rangle)(\gamma_1) = orc^1(\langle a,b,e\rangle)(\gamma_2) = orc^1(\langle a,h\rangle)(\gamma_9) = 1$, and $orc^1(x)(y) = 0$ for all other x and y. It is easy to see that orc^1 is both a basic oracle function and a standard oracle function (see Definition 3.4.8).

Given a trace, a Petri net, and a likelihood cost function, the approach in Section 4.6 computes the set of all optimal alignments between the trace and the net. Assuming that all optimal alignments have the same probability, we use this approach to construct an oracle function that returns the *same probability for all optimal alignments* between each trace in the example log and net N. Each optimal alignment has a probability of 1/(size of the set of optimal alignments set). Thus, we define an oracle function orc^A for the log and net N in Figure 6.2 such that $orc^A(\langle a,b,c,d,e,f\rangle)(\gamma_1) = 1, orc^A(\langle a,b,e\rangle)(\gamma_x) = \frac{1}{7}$

$$\gamma_x = \begin{array}{|c|c|c|c|c|c|} a & b & c & d & e & \gg \\ \hline a & b & c & d & e & f \\ t_1 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \qquad \gamma_y = \begin{array}{|c|c|} a & \gg \\ \hline a & h \\ t_1 & t_9 \end{array}$$

Figure 6.3: Left: An optimal alignment between $\sigma_x = \langle a, b, c, d, e\rangle$ and the model of Figure 6.2, **Right:** An optimal alignment between $\sigma_y = \langle a\rangle$ and the same model.

for $2 \leq x \leq 8$, $orc^A(\langle a, h\rangle)(\gamma_9) = 1$, and $orc^A(y)(z) = 0$ for all other y and z. It is easy to see that orc^A is a standard oracle function, but not a basic oracle function.

The third oracle function uses the *representatives of all optimal alignments* between any given trace and a Petri net. In Section 4.6, we showed that knot nodes can be used to group a set of similar optimal alignments. Suppose that we choose knot nodes at level 1 and the set of representative optimal alignments for the following traces: $\langle a.b, c, d, e, f\rangle, \langle a.b, e\rangle$, and $\langle a, h\rangle$, are $\{\gamma_1\}, \{\gamma_2, \gamma_4, \gamma_6\}$, and $\{\gamma_9\}$ respectively. orc^R is the oracle function constructed from the representatives such that $orc^R(\langle a, b, c, d, e, f\rangle)(\gamma_1) = 1$, $orc^R(\langle a, b, e\rangle)(\gamma_2) = orc^R(\langle a, b, e\rangle)(\gamma_4) = orc^R(\langle a, b, e\rangle)(\gamma_6) = \frac{1}{3}$, $orc^R(\langle a, h\rangle)(\gamma_9) = 1$, and $orc^R(x)(y) = 0$ for all other x and y. Note that we assign the same probability for each representative in orc^R. Hence, orc^R is a standard oracle function.

6.3 Measuring the Quality of Alignments

Given a trace and a Petri net, there are many alignments that one can construct between them as explained in the previous chapters. In Chapter 3, we showed that optimal alignments between the trace and the model yield the minimum total likelihood cost between the trace and the model. However, such minimum cost may not be sufficient to compare the quality of two arbitrary alignments. Take for example the Petri net N shown in Figure 6.2 and two traces $\sigma_x = \langle a, b, c, d, e\rangle$ and $\sigma_y = \langle a\rangle$ in the log. Optimal alignments between the two traces and net N with respect to the standard likelihood cost function are shown in Figure 6.3.

Both optimal alignments in Figure 6.3 show exactly one deviation. However, notice that γ_y is much shorter than γ_x. Intuitively, the quality of γ_x should be better than γ_y. Therefore, when comparing two alignments computed from two different traces and the same Petri net, we also take into account the length of the traces. We use the following metrics to measure the quality of alignments:

Definition 6.3.1 (Alignment quality)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $lc : (A^{\gg} \times T^{\gg}) \to I\!R$ be a likelihood cost function for movements. The quality of alignment $\gamma \in \Gamma_{\sigma,N}$ with respect to likelihood function lc is

$$aql(\gamma, N, lc) = 1 - \frac{\sum_{(x,y)\in\gamma} lc((x, y))}{lim(\pi_1(\gamma)_{\downarrow A}, N, lc)}$$

where lim is the likelihood cost limit between σ and N with respect to lc (see Definition 3.4.4). ⌟

The metric in Definition 6.3.1 takes into account the likelihood cost of movements, the length of traces, and the size of the net altogether. Let lc be the standard likelihood cost function. Sequence of transitions $\varrho = \langle t_1, t_9\rangle$ is a complete firing sequence of net

N in Figure 6.2 that yields the minimum total standard likelihood cost, i.e., ϱ is a firing sequence of N with the minimum total likelihood cost of moves on model $\sum_{t \in \varrho} lc((\gg, t)) = 2$. The quality of alignment γ_1 in Figure 6.3 is $aql(\gamma_1, N, lc) = 1 - \frac{1}{5+2} = \frac{6}{7} \approx 0.85$, while the quality of optimal alignment γ_2 is $aql(\gamma_2, N, lc) = 1 - \frac{1}{1+2} = \frac{2}{3} \approx 0.67$. By taking into account the likelihood cost of optimal alignments, trace lengths, and the size of Petri nets, the quality of two optimal alignments can be compared even if they have different lengths.

Alignments explicitly show the occurrences of invisible transitions. Thus, the alignment quality defined in Definition 6.3.1 also takes into account the likelihood cost of invisible transitions. A move on model of an invisible transition in an alignment decreases the quality of the alignment if the likelihood cost of the move is higher than 0.

An alignment between a trace and a Petri net is high likely represents the relation between the trace and the net if its quality is close to 1. In the following theory, we show that the quality of optimal alignments always falls between 0 and 1:

Theorem 6.3.2 (Optimal alignment quality range)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $\sigma \in A^*$ be a trace over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $lc : (A^{\gg} \times T^{\gg}) \rightarrow I\!R$ be a likelihood cost function for movements. For all $\gamma \in \Gamma^o_{\sigma,N,lc} : 0 \leq aql(\gamma, N, lc) \leq 1$. ⌟

Proof. Both the total likelihood cost of the alignment and the likelihood cost limit of optimal alignments are always higher than 0 (see Definition 3.4.4). Thus, $aql(\gamma, N, lc) \leq 1$. Proposition 3.4.5 shows that $lim(\sigma, N, lc)$ is an upperbound value to any likelihood cost of optimal alignment between the trace and the net. Hence, $0 \leq aql(\sigma, N, lc) \leq 1$. □

6.4 Beyond Activity Alignments

The general approach in Figure 6.1 shows that oracle functions have a crucial role in alignment-based analysis. Given a trace of an event log and a Petri net, an oracle function determines which set of alignments between the trace and the model is the best describing the deviations in the trace. Note that to this point, we only consider oracle functions that take into account the sequences of activities. As explained in Section 2.3, an event of a complex event log may have other attributes than the activity attribute. In some cases, an oracle function may also need to look at the value of these attributes.

Take for example the net Figure 6.4. The net describes the process of handling computer repair request in a reparation shop. An instance of the process starts when a front desk officer registers a computer to be repaired (register). Then, a general technician diagnoses possible problems of the computer and report his diagnostics (diagnose). Based on the diagnostics report, either a software fix (software fix) or a hardware fix (hardware fix) procedure is performed by specialist technicians. The instance ends after a book keeper archives the report and fix results (archive).

Table 7.1 shows the log of an instance of the process shown in Figure 6.4. Notice that event with id 2 assigns value “hardware” to the “Problem” data attribute. Suppose that an oracle function only consider the sequences of activities. The trace of the instance in Table 7.1 is $\sigma = \langle register, diagnose, software\ fix, hardware\ fix, archive \rangle$. Using the stan-

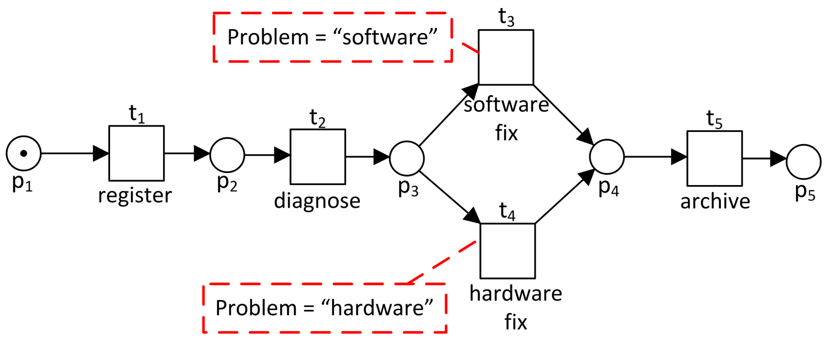


Figure 6.4: Repair process in a computer reparation shop, modeled as a Petri net annotated with data constraints.

Table 6.2: A fragment of an event log, showing an instance of the process in Figure 6.4

Event id	Properties			Data	
	Timestamp	**Activity**	**Resource**	**Problem**	...
1	20-10-2013 11:50	register	Maria	–	...
2	22-10-2013 08:10	diagnose	Roger	hardware	...
3	22-10-2013 10:04	software fix	Nadal	–	...
4	22-10-2013 10:20	hardware fix	Djokovic	–	...
5	23-10-2013 08:05	archive	Maria	–	...

dard likelihood cost function, there are two optimal alignments between the trace and the net in Figure 6.4. These two alignments are shown in Figure 6.5.

Alignments γ_1 and γ_2 show two different deviations diagnostics. γ_1 shows that activity hardware fix was performed while it is not possible according to the net in Figure 6.4. γ_2 shows that activity software fix was performed while it is not allowed according to the same net. In this example, suppose that we also take into account the "Problem" data attribute. According to the net, activity software fix can only be executed if the value of "Problem" data attribute is "software". The event log in Table 7.1 shows that the value of the attribute is "hardware". Thus, the synchronous move $(\mathit{software\ fix}, t_3)$ in the 3rd column of γ_1 is unlikely if we consider this additional information.

We can solve this problem by using a likelihood cost function that also takes into account attribute values. Instead of using a likelihood cost function that assigns cost to movements (i.e., pairs of activity names and transitions), we use a likelihood cost function that assigns cost for tuples where each tuple consists of an activity name, data value, and transition id. Suppose that the new cost function assigns cost 1 to all deviations. The

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} \mathit{register} & \mathit{diagnose} & \mathit{software\ fix} & \mathit{hardware\ fix} & \mathit{archive} \\ \hline \mathit{register} & \mathit{diagnose} & \mathit{software\ fix} & & \mathit{archive} \\ t_1 & t_2 & t_3 & \gg & t_5 \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|} \mathit{register} & \mathit{diagnose} & \mathit{software\ fix} & \mathit{hardware\ fix} & \mathit{archive} \\ \hline \mathit{register} & \mathit{diagnose} & & \mathit{hardware\ fix} & \mathit{archive} \\ t_1 & t_2 & \gg & t_4 & t_5 \end{array}$$

Figure 6.5: Two optimal alignments between trace $\sigma = \langle \mathit{register}, \mathit{diagnose}, \mathit{software\ fix}, \mathit{hardware\ fix}, \mathit{archive} \rangle$ and the net shown in Figure 6.4.

likelihood cost of γ_1 is 2 because there is a deviation for performing activity hardware fix and data constraint violation when performing activity software fix. Using the same likelihood cost, the cost of γ_2 is 1 because the only deviation in γ_2 is the move on log in the second column of the alignment. Hence, γ_2 is the optimal alignment.

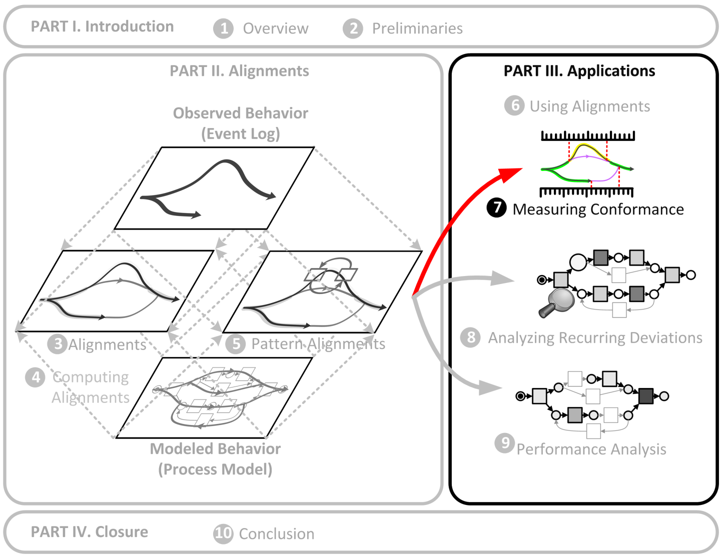
Given a complex trace, a Petri net, and a modified likelihood cost function as mentioned before, the computation of an optimal alignment that takes into account attributes other than the activity attribute can be performed using the same approach as mentioned in Chapter 4. Note that this potentially increases the complexity of the approach dramatically. It may be that many combinations of data values need to be explored by the $\mathbb{A}^\star$ algorithm. To overcome possible performance issues, in cases where the cost of control-flow deviations is much higher than the cost of data constraint violations, it may make sense to divide the alignments computation in multiple phases. In the first phase, we compute an optimal alignment by only taking into account the sequence of activities as mentioned before in Chapter 4. The result of the first phase is then filtered by taking into account data violations. The techniques to measure conformance between complex traces and Petri net with data extensions use such strategies to improve the overall computation time [44, 45].

6.5 Conclusion

Given a trace and a Petri net, the notion of alignments offers a robust approach to relate occurrences of activities in the trace to transitions in the net. In this chapter, we explained a general approach to use alignments for various types of analysis. We showed the role of oracle functions and how they can be constructed using the approaches explained in Chapter 4. We also sketched possible extensions of the approaches to take into account data information in event logs. Furthermore, we proposed a metric to measure the quality of an alignment between a trace and a Petri net that takes into account the total likelihood cost of movements, the trace length, and the size of the net altogether. In the remainder of this thesis, we assume the existence of oracle functions without explicitly mentioning the approach used to obtain them.

Chapter 7

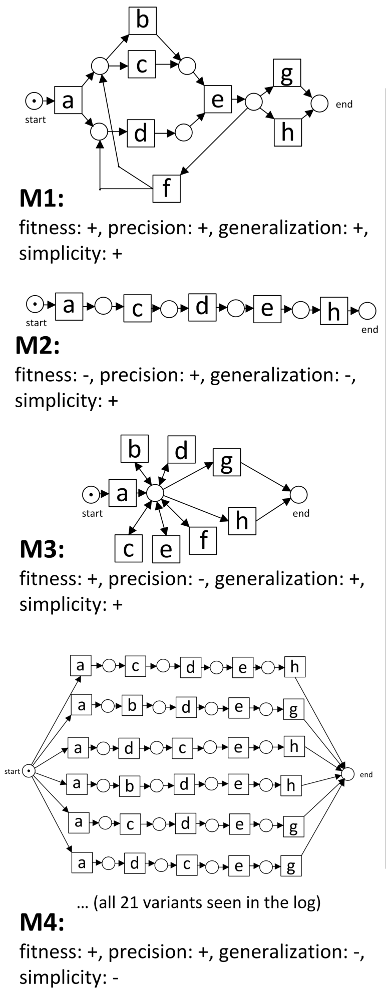
Measuring Conformance

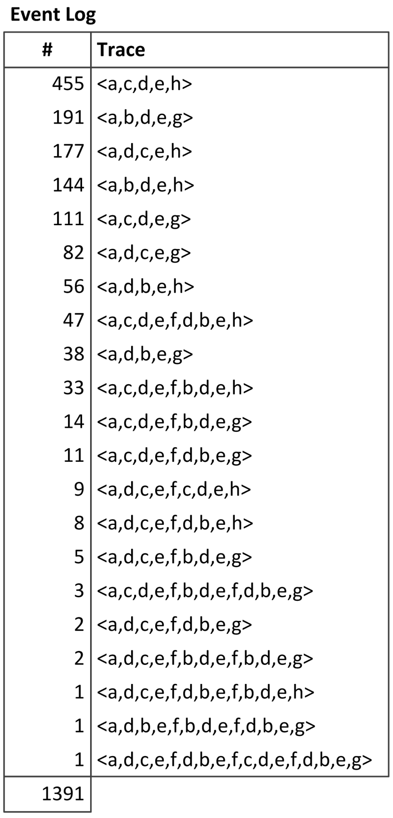


7.1 Introduction

Most Business Process Management (BPM) efforts start from process models, as they provide insights into possible scenarios [83]. Process models are used for analysis (e.g., simulation [186]), enactment [83], redesign [59], and process improvement [123, 132]. Therefore, they should reflect dominant behavior accurately. The increasing availability of event data enables the application of *conformance checking* [151, 171, 177]. Conformance checking techniques compare recorded process executions in form of *event logs* with process models to quantify how "good" are the executions with respect to their models.

Conformance can be viewed along multiple dimensions: (1) fitness, (2) precision, (3) generalization, and (4) simplicity [25, 171, 177]:


start
end
M1:
fitness: +, precision: +, generalization: +, simplicity: +
M2:
fitness: -, precision: +, generalization: -, simplicity: +
M3:
fitness: +, precision: -, generalization: +, simplicity: +
... (all 21 variants seen in the log)
M4:
fitness: +, precision: +, generalization: -, simplicity: -


Event Log

#	Trace
455	<a,c,d,e,h>
191	<a,b,d,e,g>
177	<a,d,c,e,h>
144	<a,b,d,e,h>
111	<a,c,d,e,g>
82	<a,d,c,e,g>
56	<a,d,b,e,h>
47	<a,c,d,e,f,d,b,e,h>
38	<a,d,b,e,g>
33	<a,c,d,e,f,b,d,e,h>
14	<a,c,d,e,f,b,d,e,g>
11	<a,c,d,e,f,d,b,e,g>
9	<a,d,c,e,f,c,d,e,h>
8	<a,d,c,e,f,d,b,e,h>
5	<a,d,c,e,f,b,d,e,g>
3	<a,c,d,e,f,b,d,e,f,d,b,e,g>
2	<a,d,c,e,f,d,b,e,g>
2	<a,d,c,e,f,b,d,e,f,b,d,e,g>
1	<a,d,c,e,f,d,b,e,f,b,d,e,h>
1	<a,d,b,e,f,b,d,e,f,d,b,e,g>
1	<a,d,c,e,f,d,b,e,f,c,d,e,f,d,b,e,g>
1391	

Figure 7.1: An event log L and four Petri nets M_1, M_2, M_3, and M_4 [177]

- **Fitness** measures the extent to which process models can reproduce the traces recorded in the log. Take for example the log and process models shown in Figure 7.1. Models M_1, M_3, and M_4 can reproduce all traces in the log, while model M_2 can only reproduce one variant of traces in the log (i.e., trace $\langle a, c, d, e, h \rangle$. Therefore, the *fitness* values of M_1, M_3, and M_4 are higher than M_4. Furthermore, since all models except M_2 can reproduce all traces in the log, the *fitness* values of M_1, M_3, and M_4 are the same.
- **Precision** penalizes a process model for allowing behavior that is unlikely given the observed behavior in the event log. Model M_3 and M_4 in Figure 7.1 are examples of models with extremely different precision values. Model M_4 starts with a choice and it enumerates all traces in the log. Therefore, it does not allow more behaviors than the ones logged. Thus, the precision value of M_4 is high. In contrast, after a firing of transition a model M_3 allows for simply any activity without imposing any ordering constraints. A model that exhibits such a behavior is often called a “flower” model [171]. The precision value of a flower model is low, as it does not provide any insights on how processes should be performed.

- **Generalization** evaluates the extent to which process models are able to reproduce future behavior. In general, a process model should not restrict behavior to just what is observed in the log. Generalization addresses the problem that a very specific model like M_4 may be generated whereas it is obvious that the log only holds example behavior. A model that is not general enough explains the particular sample log, but it is unlikely that another sample log of the same process can be explained well by the current model, i.e., it is unlikely that the second sample log fits well. For example, model M_4 is less general than M_1 because it models each unique trace as a unique path in the model. If another log of the same underlying process is recorded, it is unlikely that the model can explain the behavior of the log well.
- **Simplicity** penalizes models that are unnecessarily complex. This dimension is inspired by the Occam's Razor which states: *"Non sunt entia multiplicanda oracter necessitatem"*, i.e., hypotheses should not be multiplied without reason [165]. For example, model M_1 is simpler than M_4 as it contains fewer places, transitions, and arcs. Unlike the other conformance metrics, simplicity can be measured without taking into account observed behavior of process executions. For example, a Petri net is simpler than other Petri nets that allow the same set of traces if the former has less number of duplicate/invisible transitions and implicit places [29,59,107,151]. In [29], the simplicity of a Petri net is measured based on the number of activities that it represents and the number of control-flow it has. Other simplicity metrics, e.g., the one proposed in [25], also take into account the size of both process models and event logs to measure simplicity of process models. Simpler models are often more understandable and less erroneous than complex ones [108].

In this chapter, we show how alignments can be used to measure the *fitness*, *precision*, and *generalization* conformance dimensions. Next, we explain some related work with conformance checking (Section 7.2). Alignment-based conformance metrics are explained in Section 7.3. Section 7.4 provide some experimental results to validate the metrics and discussions. Section 7.5 concludes this chapter.

7.2 Related Work

In literature, there are various approaches to measure conformance between an event log and a process model. In the area of process mining, conformance checking is important to measure the quality of discovered process models with respect to event logs [148]. Thus, many conformance checking techniques are proposed as a part of process discovery techniques. In the following subsections, we discuss related work in each dimension of conformance.

7.2.1 Fitness

Fitness is arguably the most important dimension of conformance. Many approaches in literature are related to this particular dimension, e.g., [6, 8, 16, 25, 47, 68, 71, 114, 151, 152, 171, 177, 204]. Many of the approaches to relate the observed to the modeled behavior in Chapter 3 address this conformance dimension. Weijters et al. [204] propose the PM metric to quantify the fitness between event logs with respect to process models discovered from the logs. Given an event log and a process model, the metric measures the ratio of traces in the log that can be replayed correctly to the size of the log. This metric is improved by a finer-grained metric CPM that measures the ratio of events that are successfully parsed to all logged events. Although the latter improves the robustness of the former metric, it may allow for some extra behavior that is not allowed according to the original model.

Given an event log and a discovered process model from the log, Greco et al. [71] propose an approach to measure fitness based on the percentage of enactments of the discovered model that also appear as traces in the log. However, computing the metric requires exhaustive enumerations of all allowed paths by the model. Even for small process models, it might be impossible to determine all traces that are compliant with the model (e.g., if the model allows loop). Process discovery approaches that are based on genetic algorithms rely on conformance measurements to discover process models from event logs. Medeiros et al. [47] proposed a fitness metric called completeness ($PF_{complete}$) that is similar to the token-based fitness [151]. Given a log and a Petri net, the metric takes into account the number of missing and remaining tokens to replay all traces in the log on the net. However, this metric has the same problem as CPM. The addition of missing tokens may allow extra behavior that cannot be performed in the original model. Rozinat et al. [151] proposed a similar metric to measure fitness between a Petri net and an event log and proposed some extensions to deal with duplicate and invisible transitions in the net. However, this metric inherits the same problem as the $PF_{complete}$ due to addition of missing tokens.

Banescu et al. [16] quantifies fitness between a trace and a process model by taking the minimum Levenshtein distance between the trace and all traces allowed by the model. Furthermore, the distance is weighted with the severity of deviations for each misalignment between the trace and the model. However, as discussed in Chapter 3, the approach has a problem to deal with models that allow for infinitely many traces. Cook et al. [39] propose two metrics SSD and NSD to quantify fitness between an event stream and a model. Both metrics take into account the total cost of insertion/deletion to make the stream parse-able according to the model. SSD is the ratio of the total cost to a maximum cost if all events in the stream are deleted/inserted. NSD can be viewed as a variant of SSD where the same type of deviations that occur consecutively is only accounted once. These two metrics are most similar to the one we will propose in Section 7.3.1 among all metrics mentioned in this section. However, both metrics are not guaranteed to provide values between 0 and 1.

Some fitness metrics are based on strong assumptions on either event logs or process models. Goedertier et al. [68] propose the behavioral recall metric to measure the fitness between an event log and a process model. Given an event log and a process model, the behavioral recall between them is the ratio between the number of events that can be replayed correctly to the total number of events in the log. This metric is similar to the CPM metric of [204]. However, it is assumed that the event log shows the complete behavior of the modeled process. This means that an event log of a small model consists of 8 interleaving activities must contain at least $8! = 40{,}320$ uniquely different traces that contain all permutations of the activities. Similarly, the behavioral profile of Weidlich et al. [202] assumes that there are no loops in process models. These assumptions clearly limit the practical applicability of the approaches. Furthermore, a behavioral profile of a model may allow some extra behavior that is not allowed according to the original process model. Thus, a deviation in the log may remain unidentified. Van der Aalst [171] proposes fitness checking between an event log and a process model based on the so-called footprint matrices. However, this approach is only meaningful if the log is complete with respect to the "directly follow" relation between activities. Note that this is a much weaker assumption than expecting all possible traces to happen.

It is also worth mentioning that fitness is not only defined for imperative process models. De Leoni et al. [43] propose an approach to quantify the fitness between a Declare model (i.e. a declarative process model) and an event log. The fitness takes into account the cost of performing activities in the log that are not allowed according to the model and vice versa. A perfectly fitting log has a fitness value of 1. This fitness value decreases towards 0 as the number of deviations increases.

7.2.2 Precision

Compared to the number of approaches that address the fitness dimension of conformance, not many approaches in literature address the *precision* dimension. Given a log and a process model, Rozinat et al. [151] propose the (advanced) behavioral appropriateness metric that compares all pairwise relations between activities in the log to the pairwise relations of activities in the model. However, computing such pairwise relations requires an exhaustive simulation of the model which might be not feasible in practice [48]. Medeiros et al. [47] propose the $PF_{precise}$ metric to compare the precision of a process model (represented in the form of a causal matrix) to a set of process models with respect to an event log. Given a process model and an event log, $PF_{precise}$ value of the model consider the total number enabled activities in all visited states of the model when replaying the log on the model. In cases where the log is not perfectly fitting the model, "missing tokens" are added just like the approach in [151]. Thus, $PF_{precise}$ may show misleading results as the model may allow for new behavior during replay that is not allowed in the original model.

The behavioral specificity metric proposed by Goedertier et al. [68] uses artificial negative events to measure precision between an event log and a process model. The idea of this metric is to penalize precision for enabled negative events while replaying the positive events in the log on the model. However, as mentioned before, the strong completeness assumption on the log may limit the applicability of this approach. This problem is inherited by other approaches that also depends on artificial negative events, such as the F-measure of De Weerdt et al. [49]. vanden Broucke et al. [195] relaxed this assumption by improving the ways negative events are derived from event logs. Given an event log, some of improvements to derive negative events from the log are: (1) using dynamic windows instead of static windows for each event in the log, (2) exploit the information about parallelism and loop to enrich the event log before negative event extractions, and (3) exploiting both explicit and implicit dependencies between activities. However, many parameters need to be adjusted properly in order to obtain a set of negative events that are both correct and complete.

Munoz-Gama et al. [115] introduce the concept of escaping edges to measure precision between an event log and a process model. The approach is based on prefix automata and allows for measurement without having to explore all states of the model. The approach is extended to provide some degree of confidence to the obtained precision values [116]. However, the approach assumes that the log perfectly fitting the model and process models are deterministic. These are strong assumptions that may limit the applicability of the approach in practice. Buijs et al. [25] use the same precision metric to discover a process tree from an event log using the genetic mining algorithm. Interestingly, the work in [25] also shows that if the fitness dimension of a discovered model is not good enough, other quality dimensions only add little value as the discovered model does not describe the recorded behavior.

7.2.3 Generalization

Not many work in literature addresses the *generalization* dimension of conformance. The main difficulties with the generalization dimension is that we need to reason about unobserved behavior. In [174], van der Aalst illustrates the relation between real processes, event logs, and process models. Real processes are typically unknown and therefore we can only estimate them from event logs. However, an event log may contain behaviors that are not allowed according to the real process. Buijs et al. [25] define a generalization metric for an event log and a process tree based on the frequency of tree nodes visited according to the result of replaying the log on the process tree. If some parts of the tree are very infrequently visited then the generalization of the tree is bad. A generalization metric based on alignment is proposed in [177]. This metric will be explained further in Section 7.3.3.

7.2.4 Simplicity

Many approaches that address the *simplicity* dimension also take into account the structural properties of process models. Given an event log and a Petri net, Rozinat et al. [151] propose the simple and advanced structural appropriateness (a_S and a'_S) to measure the simplicity of the net. Interestingly, a_S takes into account the property of both event logs and Petri nets (i.e., the size of event logs and the total number of places and transitions), but a'_S only take into account the property of Petri nets (i.e., the number of transitions, the number of duplicate transitions, and the number of redundant invisible transitions). Given a process tree and an event log, Buijs et al. evaluate the simplicity of the tree by taking into account the number of duplicate nodes (i.e., similar to duplicate transitions in Petri net terminology), missing activities, the total number of tree nodes, and the number of event classes in the log. The simplicity of a process tree is low if the tree has many duplicate nodes and missing activities, but only few nodes.

The simplicity of a process model in [151] and [25] takes into account the observed behavior in an event log. However, under an assumption that an "ideal" process model is known, some simplicity metrics do not use event log at all. Given an "ideal" process model and a discovered process model from an event log, the structural precision (recall) metric of Medeiros et al. is the ratio of cells for which the causal matrix of the "ideal" model agrees with the causal matrix of the original model to the number of cells in the causal matrix of the original (discovered) model. This metric is similar to the precision and recall metric proposed by Pinter et al. [130].

Many existing metrics that quantify the complexity/understandibility of process models are also considered as simplicity metrics, e.g., [27,28,97,107,143]. Lee and Yoon [97] propose metrics to quantify the complexity of a Petri net based on its structural and dynamic (i.e., behavioral) properties. As an example, the total number of places, transitions, and arcs in a Petri net quantifies the structural complexity of the net. The maximum degree of concurrent transitions in a Petri net is an example of the dynamic properties. Notice that both metrics do not require event logs. Reijers and Vanderfesteen [143] propose some cohesion/coupling notions for workflow activities that can be used to evaluate the simplicity of workflows. Cardoso [28] proposes the Control Flow Complexity (CFC) metric based to measure the complexity of process models based on their number of AND-split, XOR-split, and OR-split semantics. Two survey papers of Cardoso et al. [29] and Laue and Gruhn [95] summarize many of the existing approaches to measure the understandability (simplicity) of process models without requiring event logs. These metrics are not just useful to quantify simplicity of process models, but also to understand errors in process models [108].

7.3 Alignment-based Conformance Checking

Given an event log and a Petri net, alignments between the traces of the log and the net provide explicit mappings between activities in the traces and transitions in the net. Such mappings are crucial in conformance measurement because occurrences of activities in the event log can be mapped in a deterministic way to the transitions in the net. In this section, we exploit such relations to quantify conformance in the fitness, precision, and generalization dimensions. An alignment-based fitness metric will be proposed in Section 7.3.1. Some metrics to measure conformance in the precision dimension are introduced in Section 7.3.2. Section 7.3.3 provides an alignment-based generalization metric.

7.3.1 Fitness

From all four dimensions mentioned in Section 7.1, fitness is the most important dimension of conformance. Given an event log and a process model, if most behavior in the model cannot be reproduced by the log then it is unlikely that the log/model is good enough to be used for any type of analysis. In most cases, other conformance dimensions are only worthwhile to be looked into if the fitness between the log and the model is relatively high.

As mentioned in Chapter 2, the execution of a case in a process is typically independent from the execution of other cases. Since the fitness dimension measures the extent to which all traces in the log can be reproduced by the model, fitness can also be measured per trace in the log. Given a trace and a Petri net, an alignment between them explicitly shows the deviations between them. In practice, deviations may have different severity. For example in an insurance claim handling process, sending too many notifications to a claimant typically have less severity than paying the claimant multiple times. Thus, we use a *severity cost function* that maps movements to their severity cost to quantify the fitness between the trace and the net.

Given an alignment between a trace in a log and a net, the fitness of the alignment is the total raw severity cost of all movements in the alignment. Note that the severity cost function can be different than the likelihood cost function used to construct the multi-set.

Definition 7.3.1 (Absolute Fitness)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $sc : (A^{\gg} \times T^{\gg}) \to I\!R$ be a severity cost function. Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N. The *absolute fitness* between L and N using oracle function orc and severity cost sc is

$$afit(L, N, orc, sc) = \sum_{\sigma \in L} \sum_{\gamma \in \Gamma_{\sigma,N}} \left(orc(\sigma)(\gamma) \cdot \sum_{(x,y) \in \gamma} sc((x,y)) \right)$$

⌟

Given an event log, a Petri net, an oracle function, and a severity cost function, the absolute fitness value shows the severity of deviations between the log and the net. If the absolute fitness value between the log and the net is 0 then the traces in the log are perfectly fitting the net. However, if the value is above zero then there exists some deviations in the log.

It is easy to see that the absolute fitness value may be higher than 1. In practice, it is often desirable to have a quantification of fitness in the range of 0 (very poor fitness) to 1 (perfectly fitting). Thus, we define a *relative fitness* metric that always provide values between 0 and 1 under the following conditions: (1) the oracle function only assigns non zero probability to optimal alignments with respect to a likelihood cost function, and (2) the severity cost function is the same as the likelihood cost function.

Definition 7.3.2 (Relative Fitness)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $sc : (A^{\gg} \times T^{\gg}) \rightarrow I\!R$ be a severity cost function. Let $orc^{sc} : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow I\!P)$ be an optimal oracle function of L and N with respect to sc. The *relative fitness* between L and N using oracle function orc^{sc} and severity cost sc is

$$rfit(L, N, orc^{sc}, sc) = 1 - \left(\frac{1}{|L|} \cdot \sum_{\sigma \in L} \sum_{\gamma \in \Gamma_{\sigma,N}} \left(orc^{sc}(\sigma)(\gamma) \cdot \frac{\sum_{(x,y)\in\gamma} sc((x,y))}{lim(\sigma, N, sc)} \right) \right)$$

⌟

We show that the relative fitness values always between 0 and 1.

Theorem 7.3.3 (Relative fitness range)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $sc : (A^{\gg} \times T^{\gg}) \rightarrow I\!R$ be a severity cost function. Let $orc^{sc} : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow I\!P)$ be an optimal oracle function of L and N with respect to cost function sc.

$$0 \leq rfit(L, N, orc^{sc}, sc) \leq 1$$

⌟

Proof. For all $\sigma \in L, \gamma \in \Gamma^o_{\sigma,N,sc} : \sum_{(x,y)\in\gamma} lc((x, y)) \leq lim(\sigma, N, sc)$ (see Proposition 3.4.5). Therefore, $0 \leq \frac{\sum_{(x,y)\in\gamma} lc((x,y))}{lim(\sigma,N,sc)} \leq 1$. $orc^{sc}(\sigma)(\gamma') = 0$ if $\gamma' \notin \Gamma^o_{\sigma,N,lc}$. Therefore, $\sum_{\gamma'\in\Gamma^o_{\sigma,N,lc}} orc^{sc}(\sigma)(\gamma') = 1$. It is easy to see that $0 \leq rfit(L, N, orc^{sc}, sc) \leq 1$ holds. □

Take for example the log and all Petri nets in Figure 7.1. Suppose that for each net, we use a basic oracle function that maps each trace in the log to an optimal alignment between the trace and the net. We use the the standard likelihood cost function to construct the oracle function and to measure the severity of deviations. The relative fitness values between the log and model M_1, M_3, and M_4 are all 1.00 (perfect) as all traces can be reproduced by the models. The relative fitness value between the log and model M_2 is 0.8, which implies that the model correctly represents about 80% of the events in the log. This value may seem high as just 455 of 1,391 traces fit completely (i.e., less than 33% of the traces can be perfectly replayed from beginning to end). However, note that all traces start with a and execute at least one d and e. Moreover, the majority of traces ends with h. This explains the high fitness value.

In some cases, the proposed metric in Definition 7.3.1 may be too restrictive as the severity cost function must be the same as the likelihood cost function. Therefore, we propose two other metrics to complement the relative fitness metric: (1) the *move log fitness (mlf)* , and (2) the *move model fitness (mmf)*.

Definition 7.3.4 (Move Log Fitness, Move Model Fitness)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $sc : (A^{\gg} \times T^{\gg}) \rightarrow I\!R$ be a severity cost function. Let $orc : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow I\!P)$ be an oracle function of L and N. The *move log fitness* (*mlf*) of log L and N using oracle function orc and severity cost sc is $mlf(L, N, orc, sc) =$

$$1 - \left(\frac{1}{|L|} \cdot \sum_{\sigma \in L} \sum_{\gamma \in \Gamma_{\sigma,N}} \left(orc(\sigma)(\gamma) \cdot \frac{\sum\limits_{\{1 \le i \le |\gamma| \ \mid \ \gamma[i] \in A \times \{\gg\}\}} sc(\gamma[i])}{\sum\limits_{\{1 \le j \le |\gamma| \ \mid \ \gamma[j] \in A \times T^{\gg}\}} sc((\pi_1(\gamma[j]), \gg))} \right) \right)$$

Similarly, the *move model fitness* value (*mmf*) of log L and N with respect to severity cost sc is $mmf(L, N, orc, sc) =$

$$1 - \left(\frac{1}{|L|} \cdot \sum_{\sigma \in L} \sum_{\gamma \in \Gamma_{\sigma,N}} \left(orc(\sigma)(\gamma) \cdot \frac{\sum\limits_{\{1 \le i \le |\gamma| \ \mid \ \gamma[i] \in \{\gg\} \times T\}} sc(\gamma[i])}{\sum\limits_{\{1 \le j \le |\gamma| \ \mid \ \gamma[j] \in A^{\gg} \times T\}} sc((\gg, \pi_2(\gamma[j])))} \right) \right)$$

┘

Regardless of the severity cost function, both move log and move model fitness return values between 0 and 1. Furthermore, they can be computed without any knowledge about the likelihood cost function used by the oracle function. Thus, we can use these metrics to compute fitness even if the oracle function does not have any likelihood cost function. The value of *mlf* implies the average ratio of the total severity cost of moves on log to the upperbound value of moves on log (without considering moves on model). Similarly, the value of *mmf* implies the average ratio of the total severity of moves on model to the upperbound total severity cost of moves on model (without considering moves on log). Thus, *mlf* and *mmf* yields the severity of deviations due to moves on log and moves on model respectively.

Take for example the log and models in Figure 7.1. We use the standard likelihood cost function to construct a basic and optimal oracle function, i.e., it yields one optimal alignment per trace. Furthermore, we use the same function to as a severity cost function and compute both *mlf* and *mmf*. *mlf* and *mmf* between the log and model M_1, M_3, and M_4 are all 1.00 (perfect). The *mlf* and *mmf* between the log and model M_2 are 0.79 and 0.83 respectively.

Both *mlf* and *mmf* consider moves on log and moves on model separately. We introduce the notion of *weighted fitness metrics* to combine both metrics.

Definition 7.3.5 (Weighted Fitness)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be an easy sound Petri net over A. Let $sc : (A^{\gg} \times T^{\gg}) \rightarrow I\!R$ be a severity cost function. Let $orc : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow I\!P)$ be an oracle function of L and N. The *weighted fitness* of log L and N using oracle function orc and severity cost sc is

$$wfit(L, N, orc, sc) = 2 \cdot \frac{mlf(L, N, orc, sc) \cdot mmf(L, N, orc, sc)}{mlf(L, N, orc, sc) + mmf(L, N, orc, sc)},$$

┘

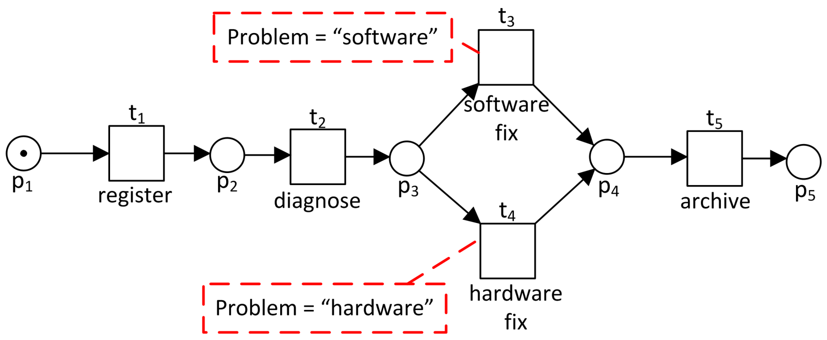


Figure 7.2: Repair process in a computer reparation shop, modeled as a Petri net annotated with data constraints.

Table 7.1: A fragment of an event log, showing an instance of the process in Figure 7.2

Event id	Properties			Data		
	Timestamp	**Activity**	**Resource**	**Problem**	**Operation Cost**	**...**
101	20-10-2013 11:50	register	Maria	–	€10	...
102	22-10-2013 08:10	diagnose	Roger	hardware	€20	...
103	22-10-2013 10:04	software fix	Nadal	–	€30	...
104	23-10-2013 08:05	archive	Maria	–	€10	...

Using the same basic and optimal oracle function as the one mentioned just before, the weighted fitness between the log and model M_1, M_3, and M_4 in Figure 7.1 are all 1.00. The weighted fitness between the log and model M_2 is $2 \cdot \frac{0.79 \cdot 0.83}{0.79+0.83} \approx 0.81$. Note that the metric always return values between 0 and 1.

The separation of the likelihood cost function that is used by an oracle function to construct alignments and the severity cost function offers some degree of flexibility to quantify fitness. Given a trace and a process model, the most likely alignment between the trace and the model may not necessarily be the one with the least severity. For example, we may determine the likelihood of an alignment based on historical information, while the severity of the alignment is determined with a fixed cost. As another example, as discussed in Section 6.4, a likelihood cost function can also be extended to consider other event attributes than just the name of activities, e.g., data attributes. Similarly, we can extend the notion of severity cost functions to also consider such extra attributes.

Take for example the repair process of a computer reparation shop in Figure 7.2. This figure was shown previously in Section 6.4. Table 7.1 shows the log of an instance of the process. Suppose that the likelihood of deviations only depends on the sequence of activities followed by the instance. An optimal alignment between the trace of the instance in Table 7.1 and the net in Figure 7.2 shows the deviations that are most likely occurred. In this example, the trace of the instance is a sequence $\sigma = \langle \mathit{register}, \mathit{diagnose}, \mathit{software\ fix}, \mathit{archive} \rangle$. Using the standard likelihood cost function, the optimal alignment between σ and the net in Figure 7.2 is shown in Figure 7.3.

Using the standard likelihood cost function, the optimal alignment in Figure 7.3 has in total zero likelihood cost. Thus, the alignment is most likely describes the relation between the trace and the net. However, note that event with id 102 in Table 7.1 assigns value "hardware" to the "Problem" data attribute. According to the annotation in

$$\gamma = \begin{array}{|c|c|c|c|} \textit{register} & \textit{diagnose} & \textit{software fix} & \textit{archive} \\ \hline \textit{register} & \textit{diagnose} & \textit{software fix} & \textit{archive} \\ t_1 & t_2 & t_3 & t_5 \end{array}$$

Figure 7.3: The optimal alignment between trace $\sigma = \langle \textit{register}, \textit{diagnose}, \textit{software fix}, \textit{archive} \rangle$ and the net shown in Figure 7.2.

Figure 7.2, activity software fix can only be performed if the value of the data attribute is "software". Thus, there is a deviation in the trace. This deviation can be captured using a severity cost function that also takes into account the values of attributes other than activity. Suppose that the severity cost function assigns costs according to the operational cost for each violating events (either for control-flow or data constraint violations). The total severity cost of alignment γ is €30, because activity software fix was performed while the value of data attribute "Problem" was "hardware" (see Table 7.1). This example clearly shows that given a trace and a Petri net, the most likely alignment between the trace and the net may not be the one with the least severity. This separation of concerns between the likelihood cost functions and the severity cost functions underlies the data conformance checking in proposed in [45]. Note that the alignment with the least severity cost may be not optimal according to the likelihood cost.

7.3.2 Precision

Given an event log and a process model, the precision of the model with respect to the log quantifies the fraction of the behavior allowed by the model which is not seen in the log. Precision of the model is high with respect to the log if the model only allows for the behavior observed in the log. If the model allows for too much behavior beyond what is observed in the log then its precision is low. Note that there is a fundamental difference between the fitness dimension and precision. While fitness values can be measured per trace, precision needs to take into account the log as a whole.

Conformance metrics need to be unidimensional [48], i.e., a conformance metric must only measure the entity that it is supposed to measure without being influenced by other entities. By definition, precision measurements assume that the behavior of the log is less or equal to the behavior allowed by the model. This is hardly the case in reality as some deviations may occur (i.e., non fitting activities may exist). For all traces in the log, the moves on model of an alignment between each trace and the model yields a complete firing sequence of the model. We exploit such alignments to construct a perfectly fitting event log from the (possibly non-fitting) log. Intuitively, a log that fits a model perfectly contains only behavior that is allowed according to the model, and nothing more.

Given a perfectly fitting log and a Petri net, we use a similar concept as the one proposed in [115, 116] to measure precision. First, we construct a precision automaton from the log that juxtaposes the behavior of the log and the behavior allowed by the net. This way, we explicitly identify the points where the net allows for more behavior than what was observed in the log, i.e., *escaping states*. Precision is then measured by taking into account the behavior exhibited by the log and the set of all identified escaping states. A precision automaton is formally defined as follows:

Definition 7.3.6 (Petri Net Precision Automaton)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. A *precision automaton* of N is a tuple $PA = (SA, SE, EA, T, ss, wgt)$ where

- SA is a set of *states*,
- SE is a set of *escaping states*,
- $EA \subseteq SA \times T \times (SA \cup SE)$ is a set of labeled directed arcs between the states,
- $ss \in SA$ is the *start state* of PA,
- $wgt : SA \to I\!R$ is a weight function that maps states to real numbers,

Furthermore, the following constraints must hold:

- Let $\varrho \in T^*$ be a sequence of transitions. If there exists a sequence $\langle(ss, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_{|\varrho|-1}, \varrho[|\varrho|], s_{|\varrho|})\rangle \in EA^*$ then $m_i \xrightarrow{\varrho} m$, i.e., a path from the start state of the automaton yields a firing sequence,
- Let $\varrho, \varrho' \in T^*$ be two firing sequences of N where arc sequence $\langle(ss, \varrho[1], sa_1), (sa_1, \varrho[2], sa_2), \ldots, (sa_{|\varrho|-1}, \varrho[|\varrho|], sa)\rangle \in EA^*$ and $\langle(ss, \varrho'[1], sa'_1), (sa'_1, \varrho'[2], sa'_2), \ldots, (sa'_{|\varrho'|-1}, \varrho'[|\varrho'|], sa)\rangle \in EA^*$ end in the same state $sa \in SA$ of PA. There exists a marking $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$ and $m_i \xrightarrow{\varrho'} m$, i.e., the two firing sequences lead to the same marking m in N.

⌟

Note that Definition 7.3.6 does not define a unique construction, i.e., there may be many precision automata per net N. Given a Petri net precision automaton, a precision value is computed as follows:

Definition 7.3.7 (Precision)
Let $A \subseteq \mathcal{A}$ be a set of activities, Let $L \in \mathbb{B}(A)$ be an event log over A. let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $PA = (SA, SE, EA, T, ss, wgt)$ be a precision automaton for L and N. Precision *prec* between L and N computed from PA is

$$prec(PA) = \frac{\sum\limits_{sa \in SA} wgt(sa) \cdot |\{(sa, t, s') \in EA \mid s' \notin SE\}|}{\sum\limits_{sa \in SA} wgt(sa) \cdot |\{(sa, t, s') \in EA\}|}$$

⌟

Definition 7.3.6 provides a generic definition on precision automata without specifying how a precision automaton must be constructed from a given event log and a Petri net. To construct such an automaton, one needs to further define the states, arcs between states, and weight function. Given an alignment multi-set, one possible way to construct a prefix precision automaton is to consider prefixes of all alignments in the multi-set as states, formalized as follows [4, 5]:

Definition 7.3.8 (Prefix Precision Automata)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N. A *prefix precision automaton* of L and N using oracle function orc is a tuple $PA = (SA, SE, EA, T, ss, wgt)$ where

- $SA = \{\varrho \in T^* \mid \exists_{\sigma \in L, \gamma \in \Gamma_{\sigma,N}}\, orc(\sigma)(\gamma) > 0 \wedge \varrho \leq \pi_2(\gamma)_{\downarrow T}\}$, i.e., the set of all prefixes of firing sequences of alignments,
- $SE = \{\varrho \cdot \langle t\rangle \in T^* \mid \varrho \in SA \wedge m_i \xrightarrow{\varrho \cdot \langle t\rangle} m' \wedge \varrho \cdot \langle t\rangle \notin SA\}$, i.e., the set of enabled transitions that are never fired,

- $EA = \{(\varrho, t, \varrho') \in SA \times T \times (SA \cup SE) \mid \varrho' = \varrho \cdot \langle t \rangle\}$,
- $ss = \langle\rangle$,
- For all $\varrho \in SA : wgt(\varrho) = \sum_{[(\sigma,\gamma) \mid \sigma \in L \wedge \gamma \in \Gamma_{\sigma,N} \wedge \varrho \leq \pi_2(\gamma)_{\downarrow T}]} orc(\sigma)(\gamma)$. Note that the weight of a state takes into account both the number of traces and the probability of alignments of the traces that "enter" the state.

┘

We show that the prefix precision automaton satisfies all of the requirements in Definition 7.3.6:

Theorem 7.3.9 (Prefix precision automaton satisfies requirements)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $orc : \{\sigma \in L\} \rightarrow ((A^{\gg} \times T^{\gg})^* \rightarrow I\!\!P)$ be an oracle function of L and N, and let $PA = (SA, SE, EA, T, ss, wgt)$ be a prefix precision automaton for L and N using oracle function orc. PA is a Petri net precision automaton.

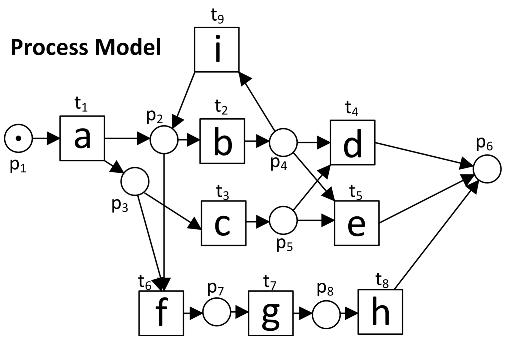
┘

Proof. According to Definition 7.3.8, there are two requirements that need to be satisfied by PA. We consider the two requirements separately.

- Let $\varrho \in T^*$ be a sequence of transitions such that $\langle(ss, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_{|\varrho|-1}, \varrho[|\varrho|], s)\rangle \in EA^*$ be a path from the start state ss. If $\varrho = \langle\rangle$ then $m_i \xrightarrow{\langle\rangle} m_i$ (the empty sequence $\langle\rangle$ is a firing sequence). If $\varrho = \varrho' \cdot \langle t \rangle$ then we know by definition that there exists $\sigma \in L, \gamma \in \Gamma_{\sigma,N}$ such that $\varrho' \cdot \langle t \rangle \leq \pi_2(\gamma)_{\downarrow T}$ and $orc(\sigma)(\gamma) > 0$. Furthermore, there exists $\varrho'' \in T^*$ such that $m_i \xrightarrow{\varrho'} m' \xrightarrow{t} m'' \xrightarrow{\varrho''} m_f$. By repeating the same procedure iteratively for ϱ', we prove that ϱ is a firing sequence, i.e., there exists $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$.
- We prove the second requirement by induction. Let $\varrho, \varrho' \in T^*$ be two sequences of transitions and let $\sigma = \langle(ss, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_{|\varrho|-1}, \varrho[|\varrho|], st)\rangle \in EA^*$ and $\sigma' = \langle(ss, \varrho'[1], s'_1), (s'_1, \varrho'[2], s'_2), \ldots, (s'_{|\varrho'|-1}, \varrho'[|\varrho'|], st)\rangle \in EA^*$ be two sequences of edges that meet at state $st \in SA$. If $\varrho = \varrho' = \langle\rangle$ then it is easy to see that firing both sequences from the initial marking leads to the same marking. By definition, $\varrho[|\varrho|] = \varrho'[|\varrho'|]$, thus $s_{|\varrho|-1} = s'_{|\varrho'|-1}$. By repeating the same procedure iteratively, we know that $\varrho = \varrho'$. Previously, we prove that $(N, m_i)[\varrho\rangle$. Thus, there exists $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$ and $m_i \xrightarrow{\varrho'} m$.

□

As shown in Definition 7.3.8, a prefix precision automaton is constructed using an oracle function. Given an event log and a Petri net, we can use at least three alternative approaches to construct an oracle function for the net and the log as discussed in Chapter 6: Using the approach to compute one optimal alignment per trace, all optimal alignments per trace, or representatives of all optimal alignments per trace. Thus, there are some alternative approaches to construct a prefix precision automaton from a given event log and a Petri net. Take for example the Petri net and log shown in Figure 7.4. Figure 7.5 shows one optimal alignment for each trace in the log. Figure 7.6 is the prefix



Event Log

Trace	Frequency
a	1
abcd	1
acbe	1
afgh	1
abibcd	1

Figure 7.4: A cancer patient handling process in a hospital and a non fitting event log.

$$\gamma_1 = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & f & g & h \\ t_1 & t_6 & t_7 & t_8 \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|c|} a & b & c & d \\ \hline a & b & c & d \\ t_1 & t_2 & t_3 & t_4 \end{array} \quad \gamma_3 = \begin{array}{|c|c|c|c|} a & c & b & e \\ \hline a & c & b & e \\ t_1 & t_3 & t_2 & t_5 \end{array}$$

$$\gamma_4 = \begin{array}{|c|c|c|c|} a & f & g & h \\ \hline a & f & g & h \\ t_1 & t_6 & t_7 & t_8 \end{array} \quad \gamma_5 = \begin{array}{|c|c|c|c|c|c|} a & b & i & b & c & d \\ \hline a & b & i & b & c & d \\ t_1 & t_2 & t_9 & t_2 & t_3 & t_4 \end{array}$$

Figure 7.5: One optimal alignments for each trace in the log and model shown in Figure 7.4. $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, and γ_5 are optimal alignments between the model in Figure 7.4 and $\sigma_1 = \langle a \rangle, \sigma_2 = \langle a, b, c, d \rangle, \sigma_3 = \langle a, c, b, e \rangle, \sigma_4 = \langle a, f, g, h \rangle$ and $\sigma_5 = \langle a, b, i, b, c, d \rangle$, respectively.

precision automaton constructed using an oracle function that yields one optimal alignment per trace as shown in Figure 7.5. Using the constructed prefix precision automaton, the precision value for the log and net in Figure 7.4 is 0.7907.

Figure 7.6 shows that there are 5 states that have some outgoing edges to some escaping states. Take for example state $\varrho = \langle t_1, t_2, t_9 \rangle$ in Figure 7.6. The sequence is a prefix of projection to model moves of γ_5, i.e., $\varrho \leq \pi_2(\gamma_5)_{\downarrow T}$, and it is not a prefix of any other projection to model of alignments in Figure 7.5. Thus, the weight of the state is the same as the number of occurrences of trace $\pi_1(\gamma_5)_{\downarrow A}$ in the log. The firing of ϱ from the initial marking $[p_1]$ of the net in Figure 7.4 leads to the marking $[p_2, p_3]$. There are three transitions enabled from the marking: t_2, t_3, and t_6. According to the set of optimal alignments in Figure 7.6, the only transition that was fired after firing ϱ is t_2. Therefore, both states $\langle t_1, t_2, t_9, t_3 \rangle$ and $\langle t_1, t_2, t_9, t_6 \rangle$ are the escaping states of state $\langle t_1, t_2, t_9 \rangle$.

The second alternative approach is to use the oracle function that yields all optimal alignments per trace. From all traces shown in Figure 7.4, $\sigma_1 = \langle a \rangle$ is the only trace in the log that has more than one optimal alignment with respect to the standard cost function. All optimal alignments between σ_1 and the model of Figure 7.4 are shown in Figure 7.7. Figure 7.8 shows a prefix precision automaton constructed from all optimal alignments between the traces and the model shown in Figure 7.4. Note that state $\langle t_1, t_2, t_3, t_5 \rangle$ is not an escaping state in Figure 7.8, but it is an escaping state in Figure 7.6. Although $\gamma_{1,5}$ in Figure 7.7 is an optimal alignment between σ_1 and the model of Figure 7.4, it is not chosen as the optimal alignment of σ_1 in Figure 7.5. The precision value for the log and net in Figure 7.4 using the automaton shown in Figure 7.8 is 0.8267. Note that the weight of the alignments of trace σ_1 is divided equally to all 5 possible optimal

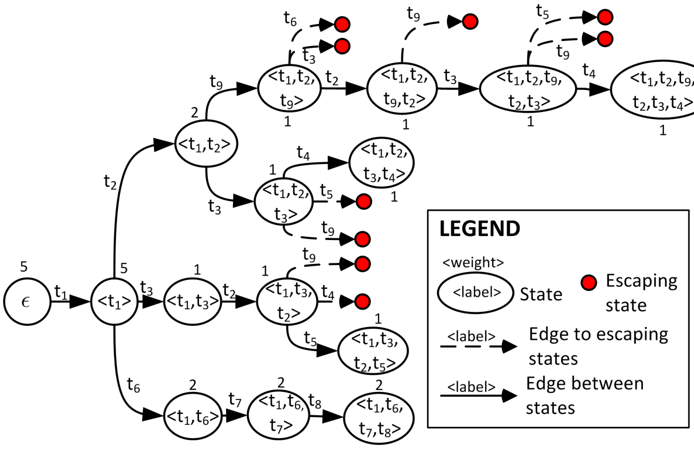


Figure 7.6: Prefix precision automaton for the log and model in Figure 7.4, using one optimal alignment per trace.

$$\gamma_{1,1} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & f & g & h \\ t_1 & t_6 & t_7 & t_8 \end{array} \quad \gamma_{1,2} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & c & b & e \\ t_1 & t_3 & t_2 & t_5 \end{array} \quad \gamma_{1,3} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & c & b & d \\ t_1 & t_3 & t_2 & t_4 \end{array}$$

$$\gamma_{1,4} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & b & c & d \\ t_1 & t_2 & t_3 & t_4 \end{array} \quad \gamma_{1,5} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & b & c & e \\ t_1 & t_2 & t_3 & t_5 \end{array}$$

Figure 7.7: All optimal alignments between trace $\sigma_1 = \langle a \rangle$ and the model shown in Figure 7.4.

alignments, i.e., the weight of each alignment is $\frac{1}{5} = 0.2$.

As a third possibility, we can use the oracle function constructed from the approach to compute representatives of all optimal alignments. Figure 7.9 shows a set of representatives of all optimal alignments between σ_1 and the model in Figure 7.4. For each representative, we also compute the represented optimal alignments (see Figure 7.7) and use them to weight the probability of the representative. A representative with higher weight has higher probability. For example, trace σ_1 has 5 optimal alignments. Since $\gamma_{1,2}$ (see Figure 7.9) represents 2 optimal alignments, its probability is $\frac{2}{5} = 0.4$. Figure 7.10 shows a prefix precision automaton, constructed using representatives of all optimal alignments. The precision value for the log and net in Figure 7.4 using the automaton shown in Figure 7.10 is 0.8027.

Note that different alternative oracle functions may yield slightly different precision results. From all three different alternatives proposed in this section, the oracle function that yields non zero probabilities only for all optimal alignments per trace provides the highest precision value because it considers more optimal alignments and therefore minimizes the number of escaping states.

In some cases, the construction used by the prefix precision automaton can be too strict. Take for example a pair of sequence $\langle t_1, t_2, t_3 \rangle$ and $\langle t_1, t_3, t_2 \rangle$ in Figure 7.10. Although each sequence in the pair is a permutation of the other sequence, they are located in different branches of the automaton as if each of them is really different than the other.

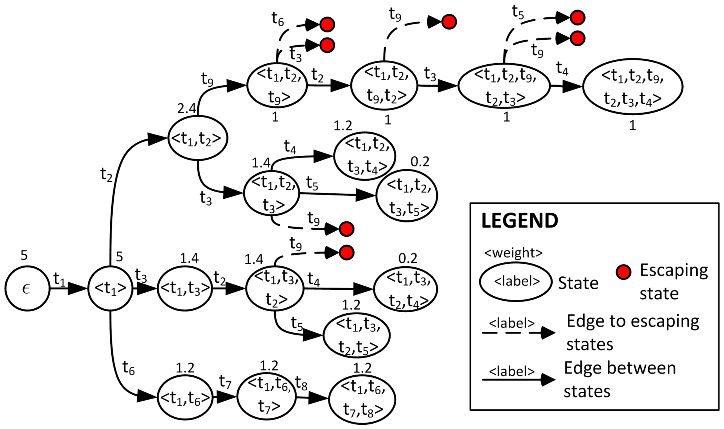


Figure 7.8: Prefix precision automaton for the log and model in Figure 7.4, using all optimal alignments per trace.

$$\gamma_{1,1} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & f & g & h \\ t_1 & t_6 & t_7 & t_8 \end{array} \textit{ representing 1 optimal alignment } (\{\gamma_{1,1}\} \textit{ in Figure 7.7})$$

$$\gamma_{1,2} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & c & b & e \\ t_1 & t_3 & t_2 & t_5 \end{array} \textit{ representing 2 optimal alignments } (\{\gamma_{1,2}, \gamma_{1,5}\} \textit{ in Figure 7.7})$$

$$\gamma_{1,3} = \begin{array}{|c|c|c|c|} a & \gg & \gg & \gg \\ \hline a & c & b & d \\ t_1 & t_3 & t_2 & t_4 \end{array} \textit{ representing 2 optimal alignments } (\{\gamma_{1,3}, \gamma_{1,4}\} \textit{ in Figure 7.7})$$

Figure 7.9: Representatives of all optimal alignments between trace $\sigma_1 = \langle a \rangle$ and the model in Figure 7.4, using knot nodes at level 1.

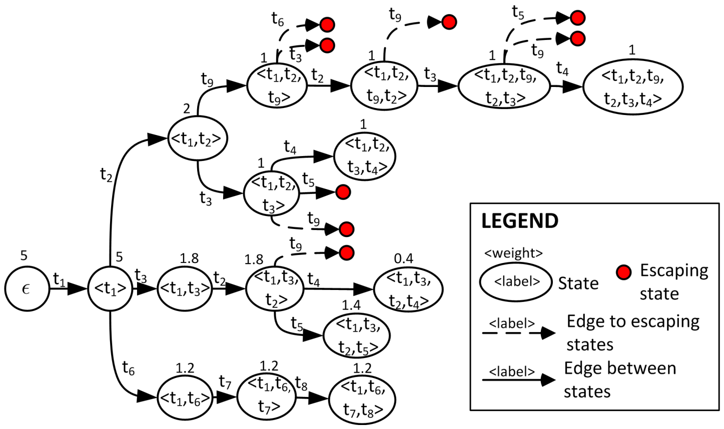


Figure 7.10: Prefix precision automaton for the log and model in Figure 7.4, using representatives of all optimal alignments per trace, knot nodes at level 1.

Therefore, we also use a state representation based on multi-set instead of prefix. This way, the two sequences are "merged" into one single state.

Multi-set Precision Automata

Given a multi-set alignment of a log and a Petri net, a *multi-set precision automaton* takes the multi-set of all prefixes of model movements in the multi-set alignment. This way, two branches that are similar in terms of the activities executed may be merged into a single state in the automaton. A formal definition of the automaton is as follows:

Definition 7.3.10 (Multi-set Precision Automata)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N. A *multi-set precision automaton* of L and N using oracle function orc is a tuple $PA = (SA, SE, EA, T, ss, wgt)$ where

- $SA = \{[t \in \varrho] \mid \exists_{\sigma \in L, \gamma \in \Gamma_{\sigma,N}, \varrho \in T^*}\, orc(\sigma)(\gamma) > 0 \wedge \varrho \leq \pi_2(\gamma)_{\downarrow T}\}$, i.e., the set of all multi-set of prefixes of firing sequences of alignments,
- $SE = \{[t'' \in (\varrho \cdot \langle t \rangle)] \notin SA \mid t \in T \wedge \varrho \in T^* \wedge [t' \in \varrho] \in SA \wedge m_i \xrightarrow{\varrho \cdot \langle t \rangle} m\}$, i.e., the set of multi-set of enabled transitions that are never fired,
- $EA = \{(b, t, b') \in SA \times T \times (SA \cup SE) \mid b' = b \uplus [t]\}$,
- $ss = []$,
- For all $b \in SA$:

$$wgt(b) = \sum_{\sigma \in L, \{\gamma \in \Gamma_{\sigma,N} \ \mid \ \exists_{\varrho \in T^*} \ \varrho \leq \pi_2(\gamma)_{\downarrow T} \wedge [t \in \varrho] = b\}} orc(\sigma)(\gamma)$$

⌟

Similar to the prefix precision automata, we prove that the multi-set automata satisfies all requirements of precision automata in Definition 7.3.6. We use a well-known Petri net theory that shows for a Petri net and a pair of firing sequence of the net for which one sequence is a permutation of the other, the pair leads to the same marking. The following result can be found in many textbook on Petri nets, e.g., [55].

Lemma 7.3.11 (Firing sequences with same transition occurrences lead to the same marking)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $\varrho, \varrho' \in T^*$ be two firing sequences of N such that for all $t \in T : \varrho(t) = \varrho'(t)$. There exists a marking $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$ and $m_i \xrightarrow{\varrho'} m$. ⌟

Proof. See for example the proof of Marking Equation Lemma in [55]. □

We use Lemma 7.3.11 to prove that multi-set precision indeed satisfies the requirement shown in Definition 7.3.6.

Theorem 7.3.12 (Multi-set precision automaton satisfies requirements)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N. Let $PA = (SA, SE, EA, T, ss, wgt)$ be a multi-set precision automaton for L and N using oracle function orc. PA is a Petri net precision automata. ⌟

Proof. The proof for this theorem is similar to the proof of Theorem 7.3.9. There are two requirements that need to be satisfied by PA. We consider the two requirements separately.

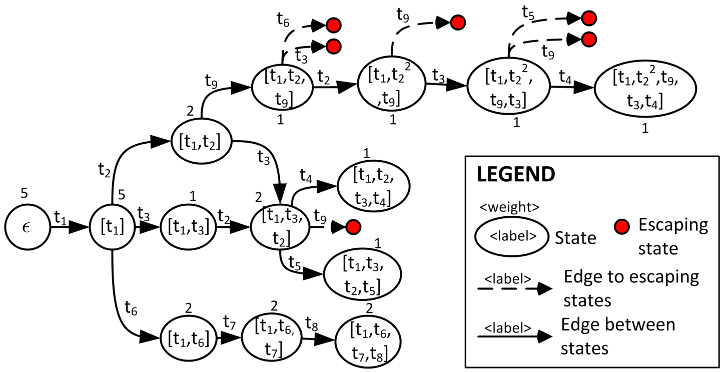


Figure 7.11: Multi-set precision automaton for the log and model in Figure 7.4, using one optimal alignment per trace.

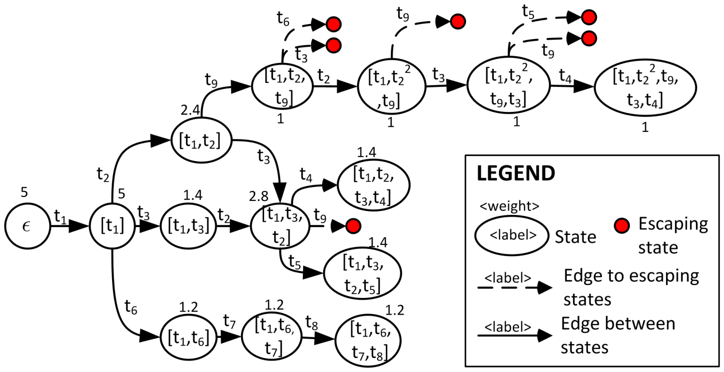


Figure 7.12: Multi-set precision automaton for the log and model in Figure 7.4, using all optimal alignments per trace.

- Let $\varrho \in T^*$ be a sequence of transitions such that $\langle (ss, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_{|\varrho|-1}, \varrho[|\varrho|], s)\rangle \in EA^*$ be a path from the start state ss. If $\varrho = \langle\rangle$ then $m_i \xrightarrow{\langle\rangle} m_i$ (the empty sequence $\langle\rangle$ is a firing sequence). If $\varrho = \varrho' \cdot \langle t\rangle$ then we know by definition that there exists a trace $\sigma \in L, \gamma \in \Gamma_{\sigma,N}$ such that $\varrho' \cdot \langle t\rangle \leq \pi_2(\gamma)_{\downarrow T}$ and $orc(\sigma)(\gamma) > 0$. Furthermore, there exists $\varrho'' \in T^*$ such that $m_i \xrightarrow{\varrho'} m' \xrightarrow{t} m'' \xrightarrow{\varrho''} m_f$. By repeating the same procedure iteratively for ϱ', we prove that ϱ is a firing sequence, i.e., there exists $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$.
- We prove the second requirement by induction. Let $\varrho, \varrho' \in T^*$ be two sequences of transitions and let $\sigma = \langle (ss, \varrho[1], s_1), (s_1, \varrho[2], s_2), \ldots, (s_{|\varrho|-1}, \varrho[|\varrho|], st)\rangle \in EA^*$ and $\sigma' = \langle (ss, \varrho'[1], s'_1), (s'_1, \varrho'[2], s'_2), \ldots, (s'_{|\varrho'|-1}, \varrho'[|\varrho'|], st)\rangle \in EA^*$ be two sequences of edges that meet at state $st \in SA$. If $\varrho = \varrho' = \langle\rangle$ then it is easy to see that firing both sequences from the initial marking leads to the same marking. If this is not the case, since both sequences end up in the same state st and st is a multi-set of transitions, we know that for all $t \in T$: $\varrho(t) = \varrho'(t')$ and hence there exists marking $m \in RS(N, m_i)$ such that $m_i \xrightarrow{\varrho} m$ and $m_i \xrightarrow{\varrho'} m$ (see Lemma 7.3.11).

$\square$

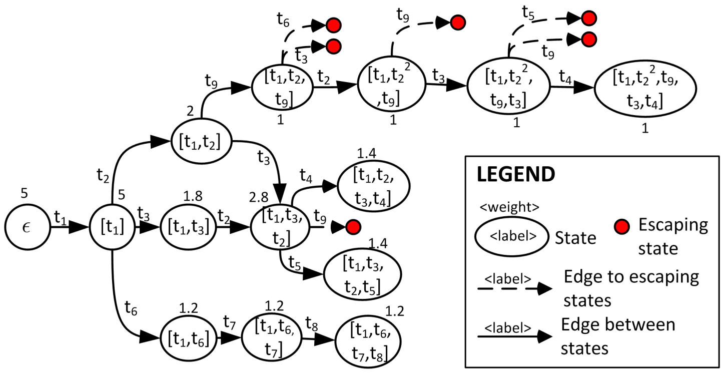


Figure 7.13: Multi-set precision automaton for the log and model in Figure 7.4, using representatives of all optimal alignments per trace.

For example, consider again the log and model shown in Figure 7.4. Figure 7.11, Figure 7.12, and Figure 7.13 are multi-set precision automata for the log and model shown in Figure 7.4, constructed by considering one optimal alignment, all-optimal alignments, and representatives of all optimal alignments respectively. Precision values computed using the automaton that consider one optimal alignment, all optimal alignments, and representatives of all optimal alignments per trace are 0.8372, 0.8267, and 0.8251 respectively. Notice that these values are typically higher than the precision values provided by prefix automata. This is because states are merged and hence there are fewer escaping edges.

Backward-Constructed Precision Automata

Given an event log, a Petri net, and an oracle function, both prefix and multi-set automata are constructed by iterating the transitions of alignments *forward*, i.e., for each alignment with non-zero probability between a trace in the log and the net, the iteration starts from the beginning of the alignment towards its end. Thus, the precision measurements are more sensitive to transitions that occur in the beginning rather than the ones that occur toward the end of alignments. Instead of constructing precision automata forward, we can also construct them backward. For each alignment with non-zero probability, we build an automaton from the end of each alignment and move backward until the beginning of the alignment is reached.

To construct a precision automaton backward using an oracle function, we reverse both the original Petri net and the set of alignments yielded by the oracle. Then, we perform the same construction of precision automata using the reversed alignments and Petri net. Recall the example event log and Petri net shown in Figure 7.4. The left side of Figure 7.14 shows the reverse of the net in Figure 7.4. The right-side of the figure shows the backward-constructed prefix precision automaton for the log and the net based on a reversed oracle function that yields one optimal alignment per trace in the log. Compare the difference between the automaton and the forward-constructed automaton shown in Figure 7.6. Precision of the log with respect to the net according to the automaton shown in Figure 7.14 is 0.8095. For each type of precision automaton that has been explained before, we can construct its backward-constructed counterpart using a similar approach as the one used to construct backward-constructed prefix precision automata.

We define the reverse of a Petri net and a reversed oracle formally as follows:

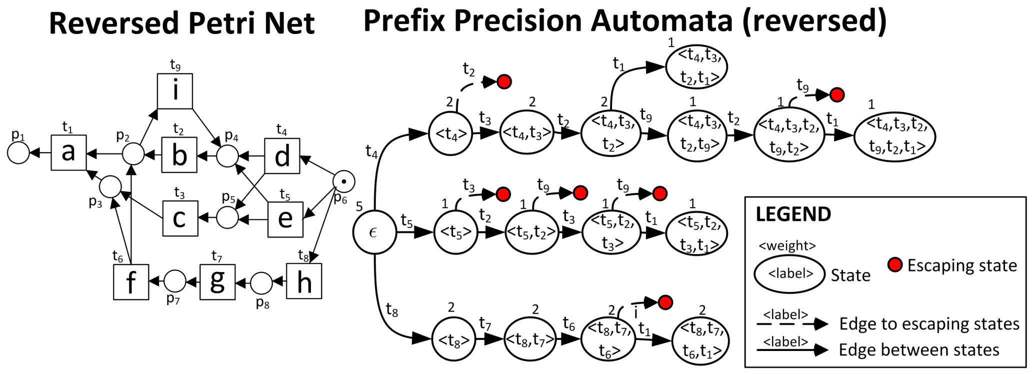

Reversed Petri Net
Prefix Precision Automata (reversed)
LEGEND
<weight>
<label> State
Escaping state
<label> Edge to escaping states
<label> Edge between states

Figure 7.14: Left: The reversed Petri net of the net shown in Figure 7.4, **Right:** Backward-constructed prefix precision automata of the log and model in Figure 7.4, using one optimal alignment per trace.

Definition 7.3.13 (Reversed Petri net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. $N' = (P, T, F', \alpha, m_f, m_i)$ is the *reverse* of N if and only if for all $(x, y) \in (P \times T) \times (T \times P) : F'(x, y) = F(y, x)$. ⌟

Definition 7.3.14 (Reversed oracle function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N.

The *reverse* of orc is a function $orc^R : \{rv(\sigma) \mid \sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ that yields probabilities for reversed alignments, such that for any reversed trace $\sigma \in \{rv(\sigma') \mid \sigma' \in L\}, \gamma \in (A^{\gg} \times T^{\gg})^*$:

$$orc^R(\sigma)(\gamma) = orc(rv(\sigma))(rv(\gamma))$$

⌟

We show that a reversed oracle function is an oracle function for a reversed log and Petri net (see Definition 7.3.13). Given an easy sound Petri net, we first show that the reverse of a firing sequence of the net is a firing sequence of the reverse of the original net.

Lemma 7.3.15 (Firing sequence in reversed Petri Net)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A, and let $N' = (P, T, F', \alpha, m_f, m_i)$ be the reverse of N. For all firing sequence $\varrho \in T^*, m_i \xrightarrow{\varrho}_N m$, let $\varrho' = \langle \varrho[|\varrho|], \varrho[|\varrho| - 1], \ldots, \varrho[1] \rangle$ be the reverse sequence of ϱ. $m \xrightarrow{\varrho'}_{N'} m_i$. ⌟

Proof. We proof this by induction. If $\varrho = \langle\rangle, m_i = m$ and thus the statement holds. Suppose that there exists $t \in T$ such that $\varrho = \varrho_1 \cdot \langle t \rangle$ and $m_i \xrightarrow{\varrho_1}_N m' \xrightarrow{t}_N m$. By definition, for all $p \in t^{\bullet}_N : m(p) = m'(p) - F(p, t) + F(t, p)$. Since $m'(p) \geq F(p, t)$, we know that $m(p) \geq F(t, p)$ and therefore $m(p) \geq F'(p, t)$. This implies that $(N', m)[t\rangle$, i.e., t is enabled at m in N'. Suppose that $m \xrightarrow{t}_{N'} m''$. By definition, for all $p \in P$: $m''(p) = m(p) - F'(p, t) + F(t, p) = m(p) - F(t, p) + F(p, t) = m'(p)$. Hence, $m \xrightarrow{t}_{N'} m'$. Iteratively, we know that $m \overset{\varrho[|\varrho|]}{\rightarrow}_{N'} m' \overset{\varrho[|\varrho|-1]}{\rightarrow}_{N'} \ldots \overset{\varrho[1]}{\rightarrow}_{N'} m_i$, thus $m \xrightarrow{\varrho}_{N'} m_i$. □

We use the lemma to prove that the reversed oracle function is an oracle function.

Theorem 7.3.16 (The reverse of an oracle function is an oracle function)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A, and Let $orc : \{\sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be an oracle function of L and N.

Let $L' = [rv(\sigma) \mid \sigma \in L]$ be the "reverse" of L, let $N' = (P, T, F', \alpha, m_f, m_i)$ be the reverse of N, and let $orc^R : \{rv(\sigma) \mid \sigma \in L\} \to ((A^{\gg} \times T^{\gg})^* \to I\!P)$ be the reverse of orc.

orc^R is an oracle function of L' and N'. ⌟

Proof. Suppose that $\sigma' \in L', \gamma' \in (A^{\gg} \times T^{\gg})^*$, and $orc^R(\sigma')(\gamma') > 0$. This implies that $orc(rv(\sigma'))(rv(\gamma')) > 0$. Thus, $rv(\gamma') \in \Gamma_{rv(\sigma'),N}$ and $m_i \xrightarrow{\pi_2(rv(\gamma'))_{\downarrow T}}_N m_f$. Using Lemma 7.3.15, we know that $m_f \xrightarrow{\pi_2(\gamma')_{\downarrow T}}_{N'} m_i$. We also know that $rv(\pi_1(\gamma')_{\downarrow A}) = \sigma' \in L'$. Hence, $\gamma' \in \Gamma_{\sigma',N'}$ (i.e., the first criteria in Definition 3.4.7 is satisfied).

Using Lemma 7.3.15, it is easy to see that for all $\sigma' \in L' : \Gamma_{\sigma',N'} = \{rv(\gamma) \mid \gamma \in \Gamma_{rv(\sigma'),N}\}$. Therefore, the following holds:

$$\sum_{\gamma' \in \Gamma_{\sigma',N'}} orc^R(\sigma')(\gamma') = \sum_{\gamma \in \Gamma_{rv(\sigma'),N}} orc(rv(\sigma'))(\gamma) = 1$$

□

We formally define backward-constructed precision automata as follows:

Definition 7.3.17 (Backward-Constructed Prefix/Multi-set Precision Automata)
Let $A \subseteq \mathcal{A}$ be a set of activities. Let $L \in \mathbb{B}(A^*)$ be an event log over A. Let N be a Petri net over A and let N' be the reverse of N. Let orc be an oracle function of L and N and let orc^R be the reverse of orc. Let $L' = [\langle \sigma[|\sigma|], \ldots, \sigma[1] \rangle \mid \sigma \in L]$ be the "reverse" of L.

The *backward-constructed prefix precision automata* of L and N is the prefix precision automata of L' and N' using oracle function orc^R. Similarly, the *backward-constructed multi-set precision automata* of L and N is the multi-set precision automata of L' and N' using oracle function orc^R. ⌟

Other than backward-constructed and forward-constructed, one can also combines both approaches by taking the average precision value of the two approaches. This way, we minimize bias in precision measurement due to the direction where we construct precision automaton.

In this subsection, we formalized an approach to measure precision for Petri nets and nets with reset/inhibitor arcs. It is easy to see that similar approaches can be applied to measure precision of process models in different modeling languages than Petri nets. Note that the backward-constructed automata exploit a specific property of Petri nets without reset/inhibitor arcs Lemma 7.3.15. This shows an example of the benefit of using Petri nets over other process modeling languages.

7.3.3 Generalization

From all four conformance dimensions, generalization is a dimension that considers both observed and unobserved behavior. A process model can be viewed as a possible encoding of a process, and an event log of the process contains some samples of the behavior of the process. Given an event log and a process model, the observed behavior in the log does not necessarily contain all possible behavior of the process. The model is said to be "overfitting" (i.e., not generalizing) if it only allows for the behavior that occurs in the log, but there is a high probability that the model is unable to explain the unlogged behavior of the process [174, 177]. Since this probability is typically unknown from the log and model, in this section we describe a general idea that can be potentially used to measure generalization.

Given an event log and a process model, we first "massage" the log to be perfectly fitting to the model such that the generalization measurement is not influenced by non-fitting traces (i.e., the measurement is unidimensional). This can be easily done using basic oracle functions: for each trace in the log, we use a basic oracle function to get an alignment and take both the moves on model and synchronous moves of the alignment to construct a perfectly fitting trace of the trace. In the remainder of this subsection, we only consider perfectly fitting logs. Given an event log that perfectly fits the model, we consider each event in the log uniquely by introducing a *massaged form* of the log as follows:

Definition 7.3.18 (Massaged Form)
Let $A \subseteq \mathcal{A}$ be a set of activities, and let $L \in \mathbb{B}(A^*)$ be an event log over A. $\mathit{split}(L) \in \mathbb{B}(A^* \times A \times A^*)$, where

$$\mathit{split}(L) = \biguplus_{\sigma \in L} [(\sigma_1, a, \sigma_2) \in A^* \times A \times A^* \mid \sigma_1 \cdot \langle a \rangle \cdot \sigma_2 = \sigma]$$

is the *massaged form* of L. Note that each event in L is mapped to precisely one (σ_1, a, σ_2) element in $\mathit{split}(L)$. ⌟

Given event log that perfectly fits a process model, we also assume that there exists a state function that maps each unique "event" in the log to a set of states of the model. We formalize such a state function as follows:

Definition 7.3.19 (State Function)
Let $A \subseteq \mathcal{A}$ be a set of activities, let $L \in \mathbb{B}(A^*)$ be an event log over A, and let N be a process model. For any $(\sigma_1, a, \sigma_2) \in \mathit{split}(L)$: $\mathit{state}((\sigma_1, a, \sigma_2)) \in S \to I\!P$ is a function that maps elements of the massaged form of L to a set of states S of the model, such that $\sum_{s \in S} \mathit{state}((\sigma_1, a, \sigma_2))(s) = 1$, i.e., the probabilities add up to one. ⌟

Intuitively, oracle functions can be used to realize such a state function. To quantify generalization between the log and the model, we look at every state of the model. Given an event e that occurs in state s of the model, we can find all events e' in the massaged log that occurred in the same state. Every event e' can be considered as an occurrence of an activity in state s. Suppose that state s is visited n times and there are w different activities observed in the state. If n is very large and w is very small then it is unlikely that a new event visiting the state will correspond to an activity not seen before in s. However, if n and w is in the same order of magnitude, then it is more likely that a new event visiting s will correspond to an activity not seen before in this state. Thus, we quantify the generalization value as follows:

Definition 7.3.20 (Generalization)
Let $A \subseteq \mathcal{A}$ be a set of activities, let N be a Petri net, and let $L \in \mathbb{B}(A^*)$ be an event log over A that is perfectly fitting N. Generalization between L and N is:

$$1 - \sum_{s \in S} \frac{\sum\limits_{(\sigma_1, a, \sigma_2) \in \mathit{split}(L)} \mathit{state}((\sigma_1, a, \sigma_2))(s)}{|\mathit{split}(L)|} \cdot \mathit{pnew}(\mathit{seen}(L, s), \mathit{visit}(L, s))$$

where

- $seen(L,s) = |\{a \mid (\sigma_1, a, \sigma_2) \in split(L) \wedge state((\sigma_1, a, \sigma_2))(s) > 0\}|$, i.e., the activities (possibly) seen for state s,
- $visit(L,s) = |[(\sigma_1, a, \sigma_2) \in split(L) \mid state((\sigma_1, a, \sigma_2))(s) > 0]|$, i.e., the number of times state s was (possibly) visited, and
- $pnew(w,n) : I\!N \times I\!N \to I\!P$ is a function that returns estimated probability that the next event visiting state s corresponds to an activity unobserved before.

⌟

We use an estimator inspired by [18] as follows: for all $w, n \in I\!N : pnew(w,n) = \frac{w(w+1)}{n(n-1)}$ if $n \geq w+2$ and $pnew(w,n) = 1$ otherwise. This estimation can be derived under a Bayesian assumption that there is an unknown number of possible transitions in all markings and the distribution of this probability follows a multinomial distribution. If $n \leq w+1$ the Bayesian analysis in [18] does not provide a transition probability as there are too few occurrences to properly estimate the probability. For $n = w+1$, $\frac{w(w+1)}{n(n-1)} = 1$ and for $n = w$, $\frac{w(w+1)}{n(n-1)} > 1$ (or undefined). Therefore, we assume the probability to be 1 if $n \leq w+1$. This is a reasonable assumption (especially for larger n). If most observations in a marking is unique, then it is likely that the next observation will also be unique [177]. The generalization value of a net with respect to an event log is close to 0 if it is very likely that a new trace will exhibit a new behavior (i.e., firing sequence) unseen before. In contrast, the value is close to 1 if it is very unlikely that a new trace will reveal new behavior.

Take for example log L and all Petri nets in Figure 7.1. $gen(L, M, orc)$ denotes the generalization of log L and model M according to Definition 7.3.20 using a basic oracle function orc to: (1) construct a perfectly fitting log form L and M, and (2) construct a state function (see Definition 7.3.19). Suppose that orc_{L,M_x} is a basic oracle function of L and model M_x where $1 \leq x \leq 4$. The generalization of L and M_1, M_2, M_3, and M_4 are $gen(L, M_1, orc_{L,M_1}) = 1.0$, $gen(L, M_2, orc_{L,M_2}) = 1.0$, $gen(L, M_3, orc_{L,M_3}) = 1.0$, and $gen(L, M_4, orc_{L,M_4}) = 0.99$. As expected, generalization value of M_1 and M_3 are high because the next trace to be observed is likely to fit into these models. Interestingly, generalization of M_2 is high because the moves on model of all optimal alignments of all non-fitting traces yield the same complete firing sequences of M_2. Since all traces yield the same complete firing sequence and therefore visit the same set of markings, the generalization metric value is high and it is unlikely that new traces will introduce new behavior. The generalization value for M_4 may be higher than expected, because $gen(L, M_4, orc_{L,M_4})$ takes the average over all fired transitions, and most transitions firing occur (by definition) in the highly frequent traces. For example, there are 455 occurrences of trace $\langle a, c, d, e, h \rangle$. For all intermediate markings between the initial and final marking there is only one possible next activity ($w = 1$) whereas these states are visited 455 times ($n = 455$). Hence $pnew(w,n) = \frac{1(1+1)}{455(455-1)} \approx 0$ for all events having such a trace resulting in a very high generalization value.

The effect becomes clear if we assume that each of the 21 possible traces occurs only once. Suppose that we construct a log L' from L such that $L' = [\{\sigma \in L\}]$. The generalization of M_1, M_2, M_3, and M_4 with respect to L' are 0.9935, 0.9952, 0.9975, and 0.1155 respectively. For this smaller log, it becomes clear that M_4 is indeed less general (i.e., more overfitting).

Table 7.2 shows the results of conformance measurements for all models and the log shown in Figure 7.1 considering the three conformance dimensions that can be mea-

Table 7.2: Conformance measurement between the log and models of Figure 7.1

Model	Log Fitness	Precision (1-align,prefix,forward)	Generalization (unique trace)
M_1	1.00	0.95	0.99
M_2	0.80	1.00	1.00
M_3	1.00	0.30	1.00
M_4	1.00	1.00	0.12

sured using alignments: fitness, precision, and generalization. Notice that most of the values corresponds to the intuition provided in Figure 7.1. The only exception is the generalization of M_2, as the log has already a fitness problem that further influences the other dimensions. This example shows that it is important to consider fitness first before analyzing the other conformance dimensions, as was also demonstrated in [25].

In Section 7.4, we provide show discussions and experimental results to evaluate the metrics proposed in this chapter. Furthermore, we use some real-life logs and models as case studies to evaluate the applicability of the metrics in real-life settings.

7.4 Evaluations

In this section, we provide experimental results and discussions to evaluate the metrics proposed in Section 7.3. In Sections 7.4.1, 7.4.2, and 7.4.3, we discuss the alignment-based fitness, precision, and generalization metrics respectively. Experimental results based on real-life logs and models are provided in Section 7.4.4.

7.4.1 Fitness

In Section 7.3.1, we proposed an approach to evaluate the fitness between a given log and a Petri net using an oracle function. The oracle function yields alignments. Thus, as discussed in Chapter 3, we tackle all possible issues due to invisible/duplicate transitions and complex control-flow pattern in the net. Furthermore, we also proposed the use of severity cost functions to quantify the fitness between them. This way, we allow different deviations to have different levels of severity. Hence, the fitness value indicates the severity of deviations between the log and the net. Note that the likelihood cost function that is used by an oracle function is not necessarily the same as the cost function that is used to measure the severity of deviations. We showed that if (1) the oracle is an optimal oracle function with respect to a likelihood cost function, and (2) the likelihood cost function is the same as the severity cost function, then the fitness value is between 0 and 1. Moreover, we also proposed the notions of move log/model fitness and weighted fitness that are guaranteed to provide values between 0 and 1. This way, we showed that the alignment-based fitness is intuitive, robust, and comparable.

As discussed in Section 7.2, many existing metrics in literature are based on strong assumptions that are often hard to satisfy in practice. Consequently, these metrics may show misleading results even for simple logs and models. Take for example the Petri net shown in Figure 7.15. The net has only two complete firing sequences: $\langle t_1, t_2, t_4\rangle$ and $\langle t_1, t_3, t_2, t_5\rangle$. Let $L = [\langle a, b, c\rangle, \langle a, b, d\rangle]$ be an event log consisting of 2 traces that

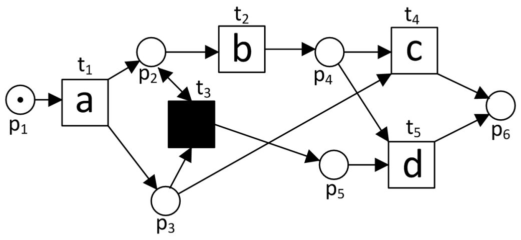


Figure 7.15: An example of process model with invisible transitions whose occurrence cannot be identified from traces of the model using the token-based replay approach [145].

a	$\gg$	b	d
a		b	d
t_1	t_3	t_2	t_5

Figure 7.16: The optimal alignment between trace $\sigma = \langle a, b, d \rangle$ and the model in Figure 7.15 using the standard likelihood cost function.

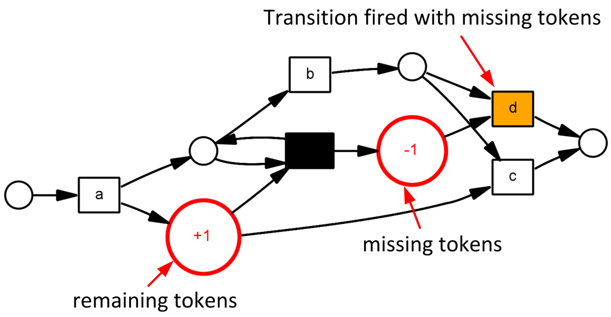
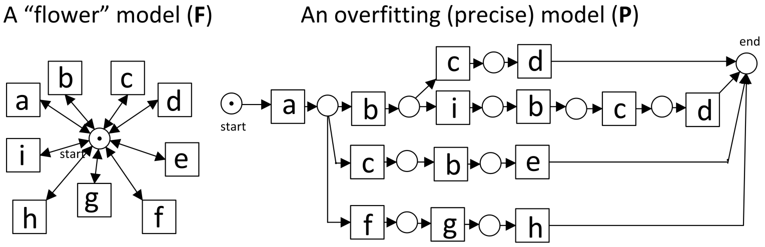


Figure 7.17: The result of replaying log $L = [\langle a, b, c \rangle, \langle a, b, d \rangle]$ and the process model in Figure 7.16 using the token-based replay approach [145] to measure fitness. Some deviations are mistakenly identified although all traces in L are perfectly fitting traces.

perfectly fits the model. We use the standard likelihood cost function to construct an optimal oracle function between L and the net. We use the same function to quantify the severity of deviations. Using the oracle function, the absolute fitness of the log and the net is 0 (all traces perfectly fit the model). Moreover, all other alignment-based fitness metrics: the relative fitness, move log fitness, move model fitness, and weighted fitness yield the value of 1. As shown in Figure 7.16, an optimal alignment between trace $\langle a, b, d \rangle$ and the model in Figure 7.15 explicitly shows invisible transition t_3.

Using the same example, the token-based fitness metric of [151] returns a fitness value of 0.9 as the token-based replay is unable to identify the occurrence of t_3 (see Figure 7.15). Similarly, the behavioral recall metric [68] provides a value of 0.83 which is a relatively low value for a log that only contains two perfectly fitting traces. The behavioral profile conformance [202] provides a perfect fitness value of 1.0 between L and the net as the net does not allow any loop. However, the limitation to models without loops might be too strong in practice.

The small example in Figure 7.15 shows that the existing fitness measurement techniques in literature may already have some problems to deal with a small-sized log and Petri net. These problems are mainly caused by the existence of invisible transitions, duplicate transitions, or complex control-flow construct (e.g., OR joins) in the model. Such



Event Log

Trace	Frequency
<a,b,c,d>	1,230
<a,b,i,b,c,d>	1,893
<a,c,b,e>	1,442
<a,f,g,h>	435

Figure 7.18: Example of an extremely imprecise (underfitting) and precise model (overfitting) for a given log.

a problem does not exist in alignment-based fitness measurement because of the explicit mapping between transitions and activities and the optimization applied to minimize the total likelihood cost.

7.4.2 Precision

In this section, we provide experimental results to evaluate the precision metrics based on various types of precision automata. First, we show that the proposed precision metrics are *unidimensional*, i.e., they are not easily influenced by problems in other dimensions of conformance. Second, we investigate the influence of different state representations of precision automata.

Evaluating Unidimensionality of Metrics

A first set of experiments was performed to evaluate the precision metrics. In particular, we measure whether the proposed precision metrics are unidimensional [48], i.e., the metrics should not be sensitive to non-fittingness of event logs. We measure precision between various logs and models whose expected values are known. Furthermore, we compare the obtained values against some existing state-of-the-art metrics for precision: etc_P [115], behavioral precision [49], and weighted behavioral precision [195].

The total time spent to compute representatives of all optimal alignments can be very long if it has to be performed for large number of logs and models (see Section 4.7). Therefore, we modify slightly the $\mathbb{A}^\star$ algorithm shown in Section 4.6 when computing representatives of all optimal alignments. Suppose that n is the best candidate node in an iteration in the algorithm, we explore all successors of n by performing the following (in sequential order): (1) synchronous moves (if possible), (2) moves on model, and (3) moves on log only if (2) can not be performed. Given an alignment, another alignment can be created by swapping a pair of move on model and move on log that follows directly one another. The modification to the $\mathbb{A}^\star$ algorithm avoids the repeated exploration of permutations of moves on model/log that occur consecutively. This implies that some paths between nodes that are constructed by the original approach may be missing and the number of represented optimal alignments computed from the exploration graph is only a lower bound instead of an exact bound as described in Section 4.5. For all experiments in this section, we use this modified algorithm to compute representatives of all optimal alignments for a given trace and Petri net.

Figure 7.18 shows an event log and two Petri nets whose expected precision values are known. The left net is the so-called "flower" net (**F**) that allows any arbitrary sequence of activities, while the right net (**P**) is a net that simply enumerates all traces in

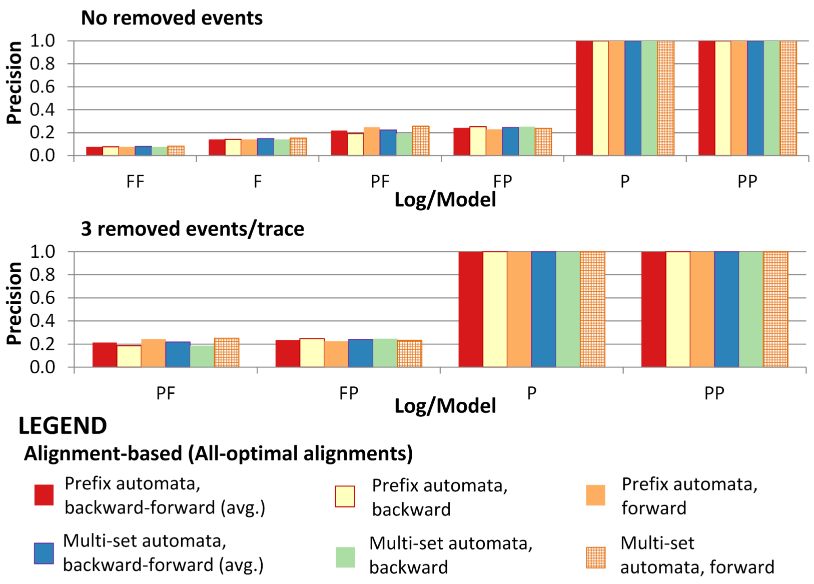


Figure 7.19: Precision values of the logs/models in Figure 7.18 and their combinations provided by alignment-based approach (i.e., computed using all optimal alignments per trace). If all behavior are observed in the original logs, all measurements are insensitive to non-fitting traces.

the log. By combining the models and log in Figure 7.18 in various ways, we have created new models whose expected precision values are between the two extremes. The two models are combined by merging the end place of one with the initially marked place of another. The merged models were named according to the name of their original models, e.g., **PF** model is the result of merging the end place of **P** with the initially marked place of **F**. The activity names in the merged models and logs were renamed before they were merged such that the original models and logs can be easily distinguished from the merged results. Precision values were measured 30 times using 30 event logs, each consisting of 5,000 traces, generated by simulating the precise model (i.e., **PP**). For the sake of completeness, we also measured the precision of the overfitting model (**P**) and the flower model (**F**) using 30 logs of 5,000 traces generated by simulating the **P** model. This way, *each log contains all the possible traces of the **P**/**PP** model*.

The top part of Figure 7.19 shows the results for the alignment-based precision metrics that are measured using an oracle function that yields all optimal alignments per trace. The experiments with oracle functions that yield one and a set of representative alignments per trace yield identical results. This result shows that by observing enough behavior in the event logs, all alignment-based metrics provide similar intuition regarding the precision of models, i.e., overfitting models have high precision values and "flower" models have low precision values. Notice that the precision values obtained using different approaches are almost the same, i.e., using different type of automata (prefix/multi-set) or different directions of constructing the automata (forward/backward).

To evaluate the robustness of the metrics against non-fitting logs, we took the models and logs from the previous experiments and created a set of unfitting logs by removing n events randomly per trace from the fitting logs. To ensure that the created logs are unfitting, only events that belong to the precise part (i.e., mapped to **P** part) are removed. We

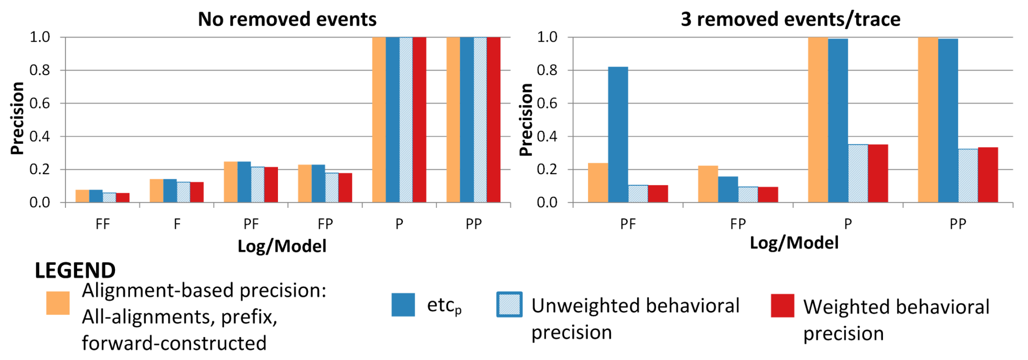


Figure 7.20: Comparison between precision values provided by alignment-based approach (i.e., computed using all optimal alignments per trace, forward-constructed prefix precision automata) and other metrics (etc_P [115], behavioral precision [49], and weighted behavioral precision [195]). Only the alignment-based approach is not sensitive to non-fitting logs/models.

measured the precision between the models and the logs and then compared it against existing metrics. We use the CoBeFra tool [194] to measure behavioral precision [49] and weighted behavioral precision [195]) and use ProM 6 to measure etc_P. The bottom part of Figure 7.19, Table 7.3, and Figures 7.20–7.22 show some of the results.

The bottom part of Figure 7.19 shows that the metrics proposed in Section 7.3 are robust to fitness problems. Even in cases where almost half of the events in all traces are removed, all alignment-based precision metrics provide a value similar to the one given for perfectly fitting traces. Figure 7.20 shows a comparison between the precision values given by alignment-based precision metrics and the values provided by other precision metrics in literature. For readability, we only show the result of one of the alignment-based metrics: the one that uses the approach to compute all-optimal alignments and a forward-constructed prefix precision automaton. Notice that in case of fitting logs, all metrics result in similar insights. In fact, the alignment-based metric shown in Figure 7.20 has the same value as the etc_P metric. However, in cases where logs are non-fitting, other metrics may show misleading precision insights. The etc_P metric provides low precision for model **PF** with respect to perfectly fitting logs (i.e., 0.25). However, the value rises to 0.82 when 3 events are removed from the logs used in the experiment, because for all non-fitting traces it ignores the rest of the traces after the first non-fitting event occurs. Similarly, both weighted and unweighted behavioral precision metrics provide lower precision values for non-fitting logs. Even for overly fitting models **P** and **PP**, both metrics provide precision values below 0.5 (i.e., indicating the models are imprecise). Because both metrics mix both perfectly-fitting and non-fitting traces in the construction of artificial negative events, the generated negative events are misleading and incorrect.

Figure 7.21 shows the influence of noise by removing some events in the logs. As shown in the figure, other than the alignment-based precision metric, precision values of all metrics may change significantly even with only one event removed from all traces. Due to the randomness of the location of removed events, the etc_P values may both increase or decrease with the presence of non-fitting traces. Both weighted and unweighted behavioral precision metrics decrease when more events are removed because incorrect artificial negative events are introduced. Note that the number of negative events tends to decrease when traces in the log show more variability because of the

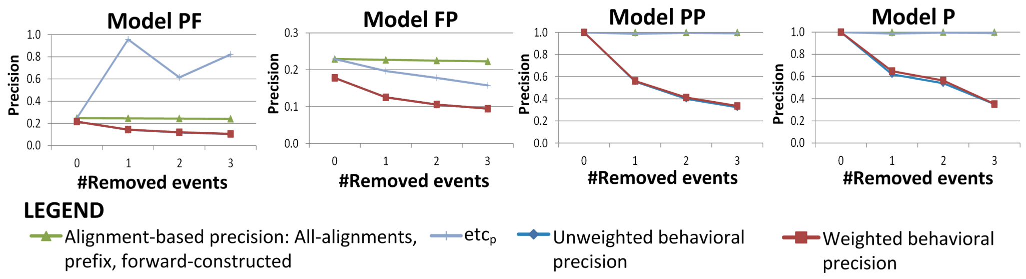


Figure 7.21: Precision values of different metrics for perfectly fitting logs and non-fitting logs created by removing some events in the logs. Only the alignment-based approach metric (i.e., computed using all optimal alignments per trace, forward-constructed prefix precision automata) is insensitive to non-fitting logs.

Table 7.3: Precision values of the **PF** model, measured using different precision automata (prefix/multi-set, forward/backward). If all behavior is observed, both one alignment and representatives of all optimal alignments per trace provide a good approximation of all-alignments per trace.

		Automata Construction Direction											
		Forward				**Backward**				**Combined**			
#Removed		**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**
Prefix	one	0.25	0.24	0.24	0.24	0.19	0.19	0.19	0.18	0.22	0.22	0.21	0.21
	rep	0.25	0.25	0.24	0.24	0.19	0.19	0.19	0.19	0.22	0.22	0.22	0.21
	all	0.25	0.25	0.24	0.24	0.19	0.19	0.19	0.19	0.22	0.22	0.22	0.21
Multi-set	one	0.26	0.25	0.25	0.25	0.19	0.19	0.19	0.18	0.22	0.22	0.22	0.22
	rep	0.26	0.26	0.25	0.25	0.19	0.19	0.19	0.19	0.22	0.22	0.22	0.22
	all	0.26	0.25	0.25	0.25	0.19	0.19	0.19	0.19	0.22	0.22	0.22	0.22

removal of events.

The set of experiments also shows some interesting insights into differences between alignment-based metrics. Table 7.3 shows the results for model **PF**. In cases where the whole behavior is recorded in event logs, precision values depend on the state representations of the automata (based on prefix/multi-set) and the directions for the automata construction. When all the possible behavior is observed, the automata constructed using one alignment or all-alignments per trace are identical. Similar results are obtained from the experiments using the other models (**P**,**F**,**FP**,**PP**,**FF**). Note the slight differences between precision values that are measured using different state representations within

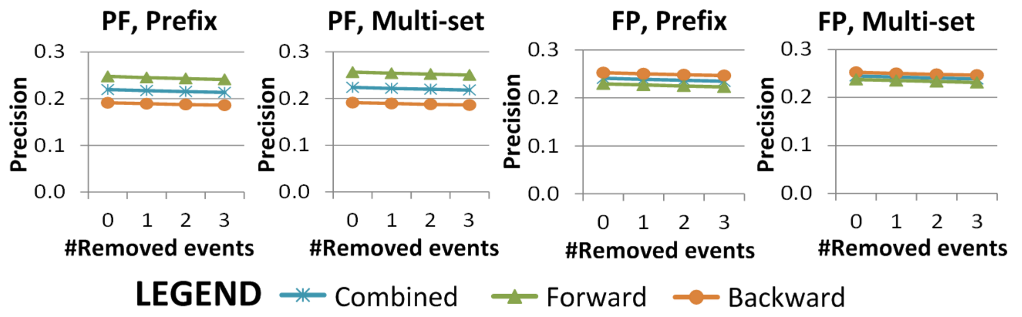


Figure 7.22: Precision values of the **PF** and **FP** using all-alignments per trace, with different precision automata (prefix/multi-set, forward/backward). Higher precision is obtained when the direction of automata construction starts with the precise part of the models.

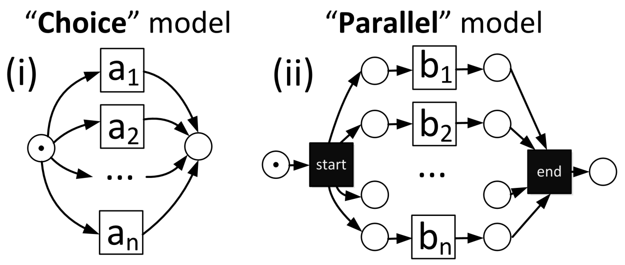


Figure 7.23: (i) A model that only allows one activity per trace, and (ii) A model that allows interleaving between all activities.

the same directions of automata construction in Table 7.3.

Table 7.3 shows that there are slight differences between precision values that are measured using different state representations and different directions in the automata construction. Figure 7.22 shows a comparison between the precision values provided by the two metrics for models **PF** and **FP**. As shown in the figure, precision values of alignment-based metrics provided by forward-constructed automata for model **PF** are higher than the values provided by backward-constructed automata for the same model, regardless of the noise level and the state representation (prefix/multi-set). In contrast, the values provided by the latter is higher than the former for the **FP** model. This shows that the position of the precise part of the models influences precision values. Precision values are higher when the direction of constructed automata starts with precise part of the process model. In this case, we clearly see the influence of direction of constructed automata (forward/backward) on precision values. To balance the influence, one of the simplest ways is to take the average between the values provided by measuring precision from automata constructed from both directions. Figure 7.22 shows that the precision values obtained by combining both values are almost similar between model **PF** and **FP**.

Thus far, non-fitting logs were created by removing activities randomly. Given a process model and a fitting trace, there are other ways to make the trace non-fitting, such as swapping some activities and add extra activities to the trace randomly. Regardless of the approach used to introduce noise, an alignment between a non-fitting trace and the model provides a good "guess" of a complete activity sequences allowed by the model that should have occurred instead of the trace. This way, precision is measured independently from other conformance metrics, e.g., the fitness metric. Other approaches investigated in this section do not explicitly handle such non-fitting logs. Hence, they are not unidimensional and may yield misleading results as shown by the experiment results presented in this section.

The Influence of State Representations

The second set of experiments were conducted to investigate the influence of state representations in alignment-based precision measurements. We use two models that, despite having the same number of activities, one allows for much more behavior than the other. The first model only allows for alternative routing (no concurrency) and is named **"Choice"** model. The second model allows for the interleaving of all activities and is named **"Parallel"** model (see Figure 7.23). For our experiments, we used models that consist of 9 activities (with invisible transitions "start" and "end" for the **"Parallel"** model).

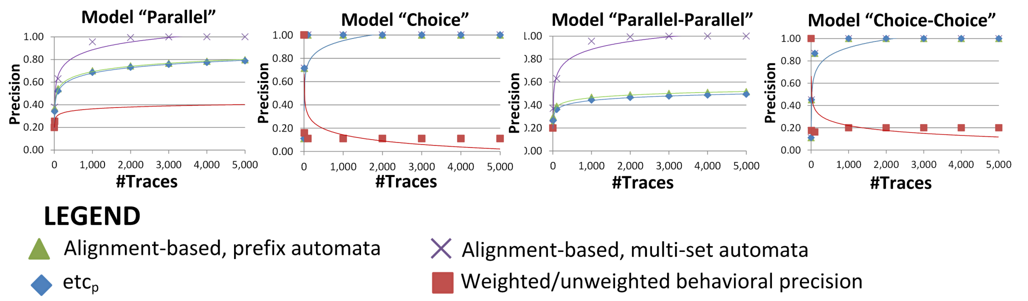


Figure 7.24: Alignment-based precision values for **"Choice"**, **"Parallel"**, **"Choice-Choice"**, and **"Parallel-Parallel"** model. For **"Parallel"** and **"Parallel-Parallel"** models, the precision values computed using multi-set precision automata are higher than the ones computed using prefix automata. For **"Choice"** and **"Choice-Choice"** models, both approaches yield the same values.

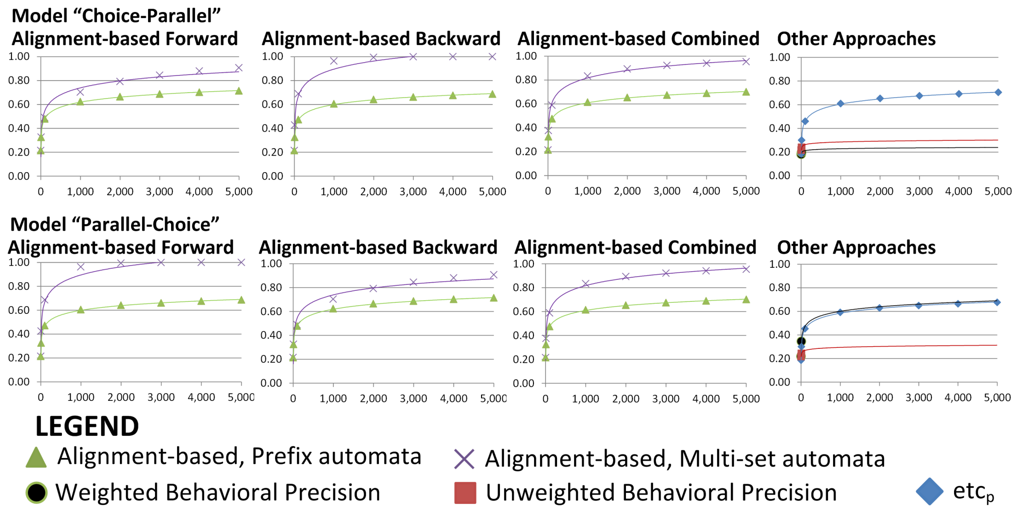


Figure 7.25: Precision values of models with combination of choice and parallel control-flow patterns. Higher precision values are obtained when the automaton is constructed starting from the part having most concurrency.

Similar to the previous set of experiments, we randomly generated perfectly fitting logs for both models with various number of traces per log and then measured their precision values. Experiments are repeated 30 times for each combination of models and number of traces per log. We conducted the same experiments with models constructed by combining the two models in sequential order (**"Choice-Choice"**, **"Choice-Parallel"**, **"Parallel-Choice"**, **"Parallel-Parallel"**). The results of the experiments are shown in Figure 7.24 and Figure 7.25.

Both Figure 7.24 and Figure 7.25 reveal that even if logs are generated from the models, all metrics require some degree of log completeness before they provide a high precision value. As expected, there is no difference between the precision values provided using multi-set and prefix precision automata for the experiments with **"Choice"** and **"Choice-Choice"** models. In contrast, the precision values provided by both approaches in the experiments with **"Parallel"** and **"Parallel-Parallel"** models are high. In theory, the minimum number of traces in an event log required to see all possible behavior of a **"Choice"** model with 9 activities is 9, while the minimum number of traces to see all possible interleaving of activities in a **"Parallel"** model is 9! = 362,880 traces. In all experiments, alignment-based precision measurements from multi-set precision automata provide the same or higher precision values with the same number of observations traces than the measurements with prefix precision automata. This experiment shows that when not all behavior has been observed in event log, precision values computed using multi-set precision automata provide the upper-bound for the ones computed using prefix precision automata.

The figure also shows that in all experiments, the etc_P values are the same as the alignment-based precision values computed using prefix automata because all traces perfectly fit the corresponding models. Interestingly, in the experiments with model **"Choice"** and **"Choice-Choice"**, both the weighted and unweighted behavioral precision metrics provide a high value (1.00) for logs with only one trace but provide very low values (below 0.2) for other logs that contain more than one trace (i.e., logs with 10, 100, 1,000, to 5,000 traces). The reason the (un)weighted behavioral precision values are so high is that the artificial negative events construction only takes into account the logged activities. When an activity in a trace of the logs is replayed to construct artificial negative events, other than the logged activity both models allow only unlogged activities (invisible transitions). Thus, no negative artificial events were constructed and therefore the precision of the models with respect to the logs is 1.00. Furthermore, the results also show that the time spent to compute alignment-based metrics is not necessarily higher than the time required to compute other existing metrics such as the (un)weighted behavioral precision. In some of the experiments with models **"Parallel"** and **"Parallel-Parallel"**, no result was obtained after 1 hour computation for (un)weighted behavioral precision while the alignment-based precision metrics were computed in less than 1 minute for each pair of model and log.

Figure 7.25 shows the precision values obtained from experiments with **"Choice-Parallel"** and **"Parallel-Choice"** models. Interestingly, the results of the experiment with **"Choice-Parallel"** model performed using forward-constructed automata (i.e., top-left of Figure 7.25) is identical to the one given by the experiment with **"Parallel-Choice"** model using backward-constructed automata (bottom-second from left of Figure 7.25). Similarly, the results of experiment with **"Parallel-Choice"** model performed using forward-constructed automata (i.e., bottom-left of Figure 7.25) is identical to the one given by the experiment with **"Choice-Parallel"** model using backward-constructed automata (top-second from left of Figure 7.25). These results show that precision values are also influenced by the location of parallel-choice constructs: precision values are higher when the automaton is constructed starting from the part of the model having most concurrent behavior. The combination precision value computed by averaging the precision values obtained from both forward and backward-constructed automata is less influenced by the construction direction as shown in Figure 7.25. As shown in the figures, the measured precision values for both **"Choice-Parallel"** and **"Parallel-Choice"** models using the combined precision values are identical. None of non-alignment-based approaches in this set of experiments managed to provide a precision value above 0.8. Note that no result was obtained after 1 hour of computation for both weighted and unweighted behavioral precision metric calculations and logs of size 1,000 traces and more.

7.4.3 Generalization

Unlike the fitness and precision metrics, the generalization metric in Definition 7.3.20 cannot be easily evaluated as it require estimations on unobserved behavior [174]. As shown by the example in Figure 7.1, given an event log and a Petri net, the proposed metric shows intuitive measurements when the number of observations are low (i.e., only considering unique traces). However, it is less intuitive when the number of observations is relatively high. This behavior can be related to the use of Bayesian assumption to provide a probability that the next visit to a visited marking in the net will reveal a new firing sequence unseen before. The assumption is valid for cases where there is an unknown number of possible transitions in all markings and the distribution of this probability follows a multinomial distribution, but in most cases there is a finite set of enabled transitions for each reachable marking.

Identifying such a probability is far from trivial. We recall the results of the work in [174] to show the challenges in measuring generalization. The work distinguishes processes, process models, and event logs as follows: A process is a function that maps traces to probabilities. A model of the process is a set of traces with probability above a predefined threshold according to the function. Event logs are multi-set of *observed* traces.

An event log L contains samples of traces of a "real" process ρ_0, while a process model N is derived from a function that maps traces to probabilities ρ_1. The "real" process ρ_0 is typically unknown. From L, one can make an estimator ρ_L of its underlying process (the process that generates the log). Generalization measurements are related to the quality of ρ_L as an estimator for ρ_0. *Generalization can be defined as the probability that the next, not yet observed, case can be replayed by the process model* [174], i.e., the probability that the next observed trace $\sigma : \sigma \in N$.

If L only contains few traces and most of them are unique, then ρ_L is a poor estimator for ρ_0. Therefore, a model that only allows for the traces in L most likely will not allow for σ. However, if L contains many traces and most of them appear many times then ρ_L is a much better estimator for ρ_0 due to the strong law of large numbers. Hence, a model allowing for the traces in L will also allow for σ.

Note that the validation of generalization remains a challenge, because the "real" process remains unknown. In most cases, we only have an event log and a process model at hand. Things are getting even more complicated as the observed behavior in the log and the modeled behavior in the model is not guaranteed to be a subset of the behavior of the "real" process.

7.4.4 Real-life Logs and Models

To evaluate the applicability of the approach to handle real life logs, we used 8 pairs of process models and logs from two different domains (see Table 7.4 and Table 7.5), where seven logs and models were obtained from municipalities in the Netherlands. In particular, we took the collections of logs and models gathered in the context of CoSeLoG project [26]. The remaining pair of log and model was obtained from a hospital in the Netherlands[1]. The logs and models from municipalities are related to different types of building permission applications, while the hospital log is related to patient handling procedure. All processes have invisible transitions, and some of the models allow loops. Table 7.4 shows an overview of the logs and models used in the experiments and the

[1]see `http://www.healthcare-analytics-process-mining.org/`

Table 7.4: The fitness values of real-life logs and models

Log	#Traces	#Events	Model		#Deviation per Trace	*afit*	*rfit*	*mlf*	*mmf*	*wfit*	Time (sec)
			#P	#T							
LM01	3,181	20,491	15	12	5.33	16,942	0.68	0.94	0.55	0.69	1.0
LM02	1,861	15,708	16	19	1.45	2,691	0.87	0.92	0.92	0.92	< 1.0
LM03	10,271	85,548	24	21	14.50	148,901	0.45	0.92	0.35	0.51	4.0
LM04	4,852	29,737	16	27	2.09	10,161	0.82	0.97	0.75	0.85	< 1.0
LM05	25,846	141,755	14	24	1.21	31,337	0.88	0.96	0.84	0.90	1.0
Bouw1	139	3,364	33	34	9.75	1,355	0.78	0.88	0.76	0.82	3.0
Bouw4	109	2,331	31	31	7.27	792	0.8	0.87	0.80	0.83	9.0
IsalaLog	77	459	26	39	0.68	52	0.94	0.94	0.95	0.94	< 1.0

Table 7.5: The generalization values of real-life logs and models

Log	#Traces	#Events	Process Model		#Deviation per Trace	*rfit*	*gen*
			#P	#T			
LM01	3,181	20,491	15	12	5.33	0.68	0.99
LM02	1,861	15,708	16	19	1.45	0.87	0.99
LM03	10,271	85,548	24	21	14.50	0.45	0.99
LM04	4,852	29,737	16	27	2.09	0.82	0.99
LM05	25,846	141,755	14	24	1.21	0.88	0.99
Bouw1	139	3,364	33	34	9.75	0.78	0.99
Bouw4	109	2,331	31	31	7.27	0.80	0.93
IsalaLog	77	459	26	39	0.68	0.94	0.98

fitness measurements results. The #Deviations per trace column indicates the cost of deviations after aligning all traces in the logs with their corresponding models. *afit*, *rfit*, *mlf*, *mmf*, and *wfit* refer to the absolute, relative, move log, move model, and weighted fitness respectively (see Section 7.3.1). We also measure the time required to compute the fitness values. To compute the fitness values, we use an oracle function that yields one optimal alignment per trace. Furthermore, we also compute the generalization between between the pairs of logs and models as shown in Table 7.5).

As shown in Table 7.4, none of the logs is perfectly fitting to the corresponding model. Some logs have more deviations than logged events (e.g., log “LM03”). Only one of the logs has a relative fitness value below 0.5 (i.e., log “LM03” has many deviations). The values of move on log/model fitness give an indication on the possible causes of deviations. A log whose *mlf* value is much higher than its *mmf* indicates that most deviations occur because of moves on model, i.e., some activities are skipped. The *mlf* values of log “LM01” and “LM03” are significantly higher than their *mmf* values because many activities in both logs are skipped. We also see that the weighted fitness values *wfit* are a good approximation of the *rfit* values. In all experiments, the computation time for all pairs of logs and models are relatively low (below 10 seconds for each pair). This result indicates the applicability of the alignment-based approach to measure the fitness of logs and models with real-life complexity.

Table 7.5 shows that the generalization values of all logs are high. These values indicate that there are sufficient behaviors observed in the logs and new traces are less likely to follow a new path in the models. However, note that the values are influenced by the fitness level of the logs.

Similarly, we measure the precision values for all logs and models and the computation time required. The results are shown in Figure 7.26 and Figure 7.27. Figure 7.26 reports precision values obtained for real-life logs and models. Only the approach based on 1-alignment provides precision values for all real-life logs and models in the experiments. The approach based on all-optimal alignments per trace had out-of-memory prob-

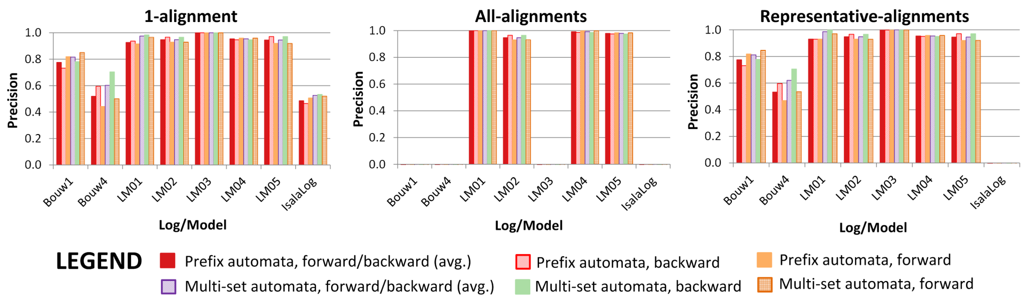


Figure 7.26: Precision values of real-life logs and models. Only the approach based on one alignment per trace manage to provide precision results for all logs/models.

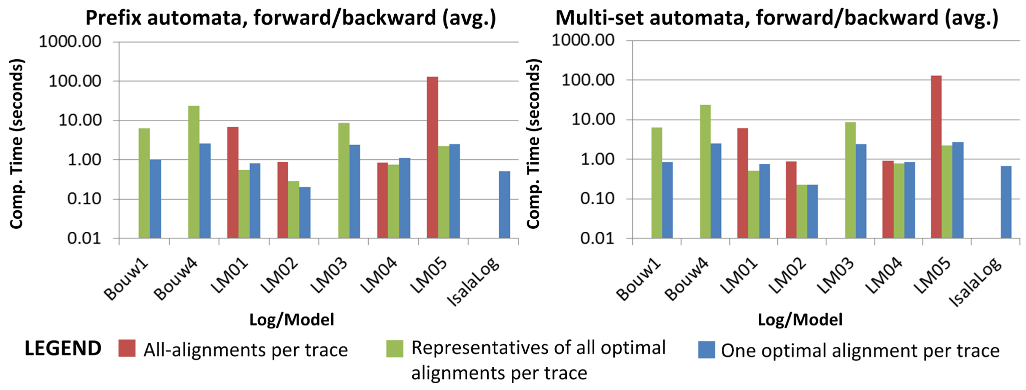


Figure 7.27: Computation time comparison of alignment-based precision measurement using combined values (from backward and forward constructed automata). Missing values are due to out-of-memory problems.

lems when dealing with relatively complex process models and logs such as "Bouw1" (33 places, 34 transitions), "Bouw4" (31 places, 31 transitions), and "LM03" (24 places, 21 transitions). Precision measurements based on representatives of all optimal alignments also had the same problems dealing with the hospital log (i.e., "IsalaLog"). The model of the hospital log is relatively small, but it contains many invisible transitions, allows loops, and allow many interleaving activities such that the size of state space required to compute even representatives of all optimal alignments is large and does not fit memory.

Nevertheless, notice the similarity of the computed precision values using all three alignments (1-align, all-align, and representatives). From all pairs of logs and models, only 2 of them have precision values below 0.7. This shows that in reality, process models are made to be relatively precise such that meaningful insights into process can be obtained. Interestingly, different insights are provided by different direction of constructing automata and state representation in the experiment with log and model "Bouw4" using both one and representative alignments per trace. The precision value of the log, computed using forward-constructed prefix automata is around 0.4 (showing slight imprecision) while the same value computed using backward-constructed prefix automata is 0.6 (showing that the log is slightly precise). This example shows the importance of balancing the influence of direction for which automata is constructed. Alternatively, this is also an indication that more observations are required to measure the precision of the particular log and model accurately.

Figure 7.27 reports the computation time required to measure precision of real-life

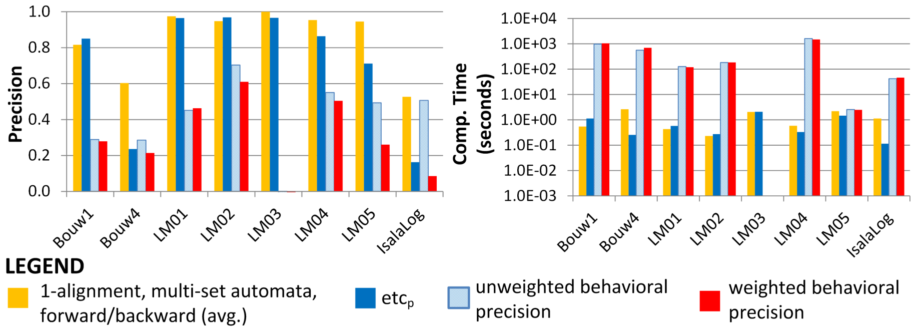


Figure 7.28: Precision values (left) and computation time (right) comparison between alignment-based precision measurements and existing precision measurements using real-life logs and models. Y-axis values on the right chart are shown in a *logarithmic* scale. Missing values indicate that no result was obtained after 1 hour of computation.

logs and models using alignment-based approach with combined precision values between forward and backward-constructed automata. The y-axis of the charts are shown using a logarithmic scale. As shown in the figure, the computation time of precision measurement with all-alignments takes much longer than the ones required by one or representative alignments. Except for the experiment with log "IsalaLog", all measurements using one optimal alignment/representative of all optimal alignments per trace were computed in less than 24 seconds. Notice the similarity between the left and right graph on the figure (except the "IsalaLog" that has out-of-memory problem in the approach with representative alignments). In fact, we obtained identical results for all other experiments with different combination of prefix/multi-set automata and directions where the automata is constructed (forward/backward). This shows that the direction to construct automata and state representations (i.e., prefix/multi-set) do not influence computation time significantly. Most time is spent on constructing alignment multi-sets.

Figure 7.28 shows a comparison of precision values and computation time between alignment-based precisions (represented by the ones computed using one alignment per trace, forward/backward constructed multi-set automata, averaged) and other approaches. In most cases, the alignment-based approach yields higher values than other approaches. The right-hand side of the figure shows that the computation time of both weighted and unweighted behavioral precision is much higher than the computation time of both the alignment-based precision and etc_P.

7.5 Conclusion

Given an event log and a process model in the form of a Petri net, we showed several alignment-based approaches to measure how good the log is with respect to the net. Regardless of the dimension of conformance that is measured, all measurement approaches in this chapter can be divided into two steps. The first step is constructing alignments from each trace in the log, and the second step is measuring conformance using the constructed alignments.

The fitness of a log to a process model is often measured based on the proportion of the observed behavior in the log that can be reproduced by the net. In this chapter, we proposed a flexible approach to measure fitness by explicitly taking into account the severity of deviations. Thus, different deviations may have different severity. The approach uses a severity cost function to quantify the severity of deviations (movements) in alignments. We showed that alignment-based fitness measurement is robust to peculiarities of process models (e.g., invisible/duplicate transitions, complex control-flow). Furthermore, we also showed that if the constructed alignments are all optimal according to a likelihood cost function and the cost function is the same as the severity cost function, the range of the fitness value is between 0 and 1.

Alignments also provide a basis to compute other metrics of conformance on different dimensions, such as precision and generalization. Many approaches to quantify precision assume perfect fitness, while this assumption is rarely being satisfied in practice. This results in unreliable precision measurements as shown in this chapter. Therefore, we have developed an approach to measure precision based on alignments. Alignments are crucial to measuring precision more accurately, especially in those cases where the log is non-fitting. Given a Petri net and an event log, we use an automaton-based approach to measure precision. Automata are mainly used as a means to juxtapose the behavior of the net with the behavior observed in the log. We showed that the choice of state representation in the construction of the automata influences the precision value obtained. Furthermore, we have identified several behavioral properties of process models that may cause a biased precision measurement. To minimize such bias, we proposed an average precision value between the automata obtained using forward and backward construction.

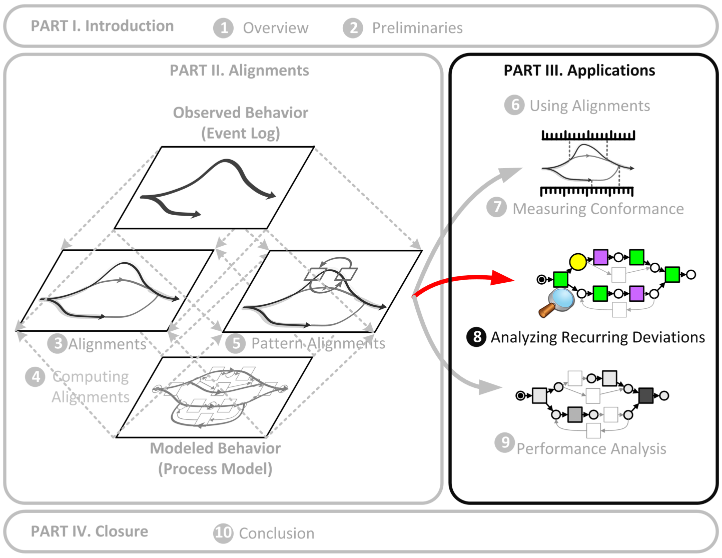
Computing all optimal alignments between a process model and an event log is computationally expensive, if not infeasible in practice. We showed that precision values based on both one optimal alignment and representatives of all optimal alignments per trace provide a good approximations for the values obtained using the all-optimal alignments approach. We also showed that the precision measurement based on representative optimal alignments provides a trade-off between computation time and metric quality, providing more diagnostics information (i.e., a lower bound of the number of optimal alignments). Nevertheless, identifying the “optimal” trade-off between computation time and rich diagnostic information remains a challenge for practical cases.

While the alignment-based fitness and precision metrics can be easily validated using either theoretical or empirical proofs, the generalization metric is hard to be justified as it reasons about unknown facts. The proposed metric provides an intuition on how to measure generalization. However, the validation of the metric remains a challenge due to the nature of the problem.

Given an event log and a process model, we stress that all dimensions of conformance must be considered together when evaluating the quality of the model with respect to the log. Fitness, however, can be viewed as the most important quality dimension as it is highly unlikely that if the model poorly fits the log, measurements on other dimensions are meaningful. This result is consistent with the discussions and results of [25, 174].

Chapter 8

Analyzing Recurring Deviations



8.1 Introduction

The dynamic nature of business environments nowadays demands organizations to be flexible and quickly adapt to changing business requirements [156, 160]. Thus, business processes need to support flexibility, e.g., by allowing alternative execution paths. Such flexibility is also called *flexibility by design* [156].

In many cases, it is impossible to predict all possible execution paths at design time. Therefore, an information system that supports a process typically allows for *deviations* from a prescribed process model. Resources are allowed to deviate temporarily from reference models at runtime without changing them and influencing the way other instances of the same process are performed. For example, in case of an emergency a patient may go straight to the operation table and skip some administrative procedures that should have been performed before surgery in normal cases, e.g., registration, appointment, etc. Nevertheless, regular patients without the same emergency status still need to follow the administration procedures mention before. This type of flexibility is categorized as *flexibility by deviation* [156].

In a process supported by a system that provides flexibility by deviation, the management of deviations directly impacts the performance of overall processes [72, 80]. Understanding the causes of deviations can help organizations to identify changes required in the process to avoid future occurrences of the deviations [72]. A frequently occurring deviation in the process – such that it can even be considered as a routine – indicates that the model of the process may need to be modified to reflect reality better [166]. Deviation analysis may reveal interesting insights that can be used to improve the way processes are conducted.

Given an event log and a process model, we are particularly interested in answering the following research questions:

- *What are the most frequently occurring deviations?* If the deviations are so frequent and can even be considered routine, one may consider improving the reference model to explicitly allow for the deviations [166],
- *In which context do deviations often occur?* Understanding the context where deviations often occur can be exploited to prevent similar deviation from occurring in the future or predicting occurrences of the same deviations while cases are still running [72, 80].

In the earlier chapters, we showed how alignments can be used to identify possible root causes of deviations in the control-flow perspective. Given an event log and a process model, in this chapter we show how alignments can be further exploited to analyze recurring deviations and provide answers to the research questions mentioned above. In Section 8.2, we first discuss related work. Alignment-based approaches that reveal different insights into deviations between the log and the model are presented in Section 8.3. Case studies to demonstrate the applicability of the approaches are reported in Section 8.4. Section 8.5 concludes this chapter.

8.2 Related Work

As shown in Chapter 3 and Chapter 7, many approaches have been proposed to check the conformance between a given process model and its recorded executions, e.g., [45, 49, 65, 68, 115, 116, 151, 202]. However, often these do not provide the desired insights. The analyst would like to see recurring deviations and the root causes of deviations.

Given an event log and a Petri net, Rozinat et al. [151] use the token replay approach to identify deviations between the traces of the log and the net. The original net is annotated with information about missing/remaining tokens and frequently visited paths, obtained from replaying the traces of the log on the net. This way, a process expert can diagnose possible root causes of deviations in a visual manner. However, in cases where deviations occur frequently, places may be flooded with missing/remaining tokens and it is impossible to find the root cause of deviations. The approach of [151] also adds annotations to show deviating relations between transitions in the net and activities in the log (e.g., always follows, always precedes, sometimes follows, and never follows). However, many of such arcs can be added such that the original net becomes unreadable if there are too many deviations.

Weidlich et al. [202] propose an approach based on behavioral profile to diagnose the root causes of deviations, given an event log and a process model. The approach uses the pairwise relations (e.g., strict order, exclusive, or interleaving order) between activities in the log and in the model to check for deviations between them. This approach is also used to monitor deviations and provides possible root causes in an online setting [203]. However, as mentioned in Chapter 3, the behavioral profile of a process model may allow for more behavior than the behavior of the original process model. Thus, the provided diagnostics may be misleading. Furthermore, the root causes of deviations are presented in the form of pairwise relations between activities without pointing out which specific activities in the log that cause the deviations.

In a case study, Swinnen et al. [162] show that the combination of process mining and data mining techniques can be used to show insights into deviations, given an event log and a process model. The proposed approach uses the Fuzzy miner technique [75] to discover an "as-is" process model from the event log. This model is then compared to the prescribed model to identify deviations. Furthermore, data mining approaches such as the association rule mining [13] are used to mine insights into frequently occurring behaviors/deviations. A similar approach is followed by Jans et al. [86] in another case study to audit a procurement process. Interestingly, in both case studies, deviations are identified by manually comparing the discovered model with the prescribed model instead of replaying the log on the prescribed model.

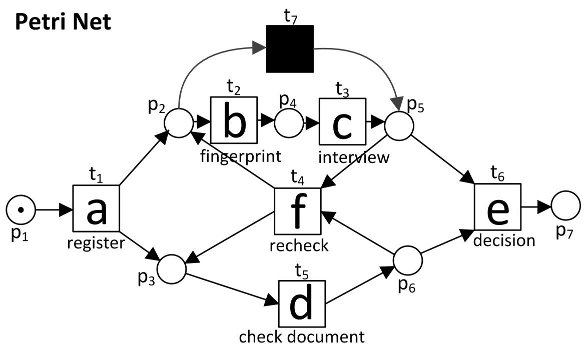
As shown in this section, not many approaches take into account both the process model and its executions in order to show insights into possible root-causes of recurring deviations. In Section 8.3, we show various ways to analyze deviations based on alignments to answer the research questions mentioned in Section 8.1.

8.3 Alignment-based Deviation Analysis

Given a trace and a process model, an alignment between them explicitly shows the location of deviations between traces in the log and the model. In this section, we show several alignment-based approaches that altogether can be used to answer all research questions mentioned in Section 8.1. As a running example, we use the event log and net shown in Figure 8.1. The figure shows a visa application handling process similar to the one shown in Chapter 5 and its non-fitting event log. The process starts when an applicant submits his visa application and registers himself at the visa center (register). An officer takes the fingerprints of the applicant (fingerprint) and then performs an interview (interview). In parallel, an administration officer checks all documents that were submitted by the applicant. A manager decides whether the application is accepted/rejected after the submitted documents are checked and the interview results are obtained (decision). If suspicious information is found, a manager may instruct a visa application officer to perform thorough checking of the applicant (recheck). Collecting fingerprints and interviews can be skipped if the fingerprints data of the applicant are already stored in the visa center central database and the applicant has been interviewed at least once. For convenience, we relabel all transitions in the net with a single letter. Figure 8.2 shows an optimal alignment for each trace of the log in Figure 8.1 and the net in the same figure with respect to the standard cost function.

Deviation analysis is often not a trivial task. Given an event log and a process model, the causes of a deviation in a trace in the log may not be the same as the cause of the same deviation in another trace in the same log. Moreover, a deviation may have multiple possible causes. In the presence of massive, dynamic, ambiguous, and perhaps

Petri Net



Event Log

ID	Trace	Frequency
1	<a,b,d,c,e>	249
2	<a,c,b,d,e>	234
3	<a,d,b,c,f,b,c,d,f,d,e>	200
4	<a,b,c,e>	115
5	<a,d,b,c,f,d,b,e>	63
6	<a,b,d,c,f,b,d,e>	63
7	<a,b,d,c,d,b,c,e>	43
8	<a,d,b,d,b,c,e>	33

Figure 8.1: A visa application handling process and its non-fitting event log.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & b & d & c & e \\ \hline a & b & d & c & e \\ t_1 & t_2 & t_5 & t_3 & t_6 \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|c|} a & c & b & d & \gg & e \\ \hline a & & b & d & c & e \\ t_1 & \gg & t_2 & t_5 & t_3 & t_6 \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} a & d & b & c & f & b & c & d & f & d & \gg & e \\ \hline a & d & b & c & f & b & c & d & f & d & & e \\ t_1 & t_5 & t_2 & t_3 & t_4 & t_2 & t_3 & t_5 & t_4 & t_5 & t_7 & t_6 \end{array} \qquad \gamma_4 = \begin{array}{|c|c|c|c|c|} a & b & c & \gg & e \\ \hline a & b & c & d & e \\ t_1 & t_2 & t_3 & t_5 & t_6 \end{array}$$

$$\gamma_5 = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & d & b & c & f & d & b & \gg & e \\ \hline a & d & b & c & f & d & b & c & e \\ t_1 & t_5 & t_2 & t_3 & t_4 & t_5 & t_2 & t_3 & t_6 \end{array} \qquad \gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & b & d & c & f & b & d & \gg & e \\ \hline a & b & d & c & f & b & d & c & e \\ t_1 & t_2 & t_5 & t_3 & t_4 & t_2 & t_5 & t_3 & t_6 \end{array}$$

$$\gamma_7 = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & b & d & c & \gg & d & b & c & e \\ \hline a & b & d & c & f & d & b & c & e \\ t_1 & t_2 & t_5 & t_3 & t_4 & t_5 & t_2 & t_3 & t_6 \end{array} \qquad \gamma_8 = \begin{array}{|c|c|c|c|c|c|c|} a & d & b & d & b & c & e \\ \hline a & d & b & & & c & e \\ t_1 & t_5 & t_2 & \gg & \gg & t_3 & t_6 \end{array}$$

Figure 8.2: An optimal alignment for each trace in the log of Figure 8.1 and the net in the same figure with respect to the standard cost function. γ_n is an optimal alignment between trace with id n with the model in Figure 8.1.

even conflicting data, visual analytics tools and techniques can be used to synthesize information and derive insights [164]. Visual analytic techniques combine information visualization with other techniques from data analysis (e.g., data mining) in order to fully exploit humans' capacity to perceive, understand, and reason about complex and dynamic data and situation.

Due to limitations with respect to perceiving data, analysis is often performed in an explorative manner. Given a set of data, it is highly unlikely that there exists a single visualization that provides all desired information "hidden" within the data. In most cases, one needs to use multiple visualizations to obtain insights based on a given set of data. Given an event log, a Petri net, and the corresponding alignments, we construct three different visualizations of the alignments to obtain insights into deviations. Each of the visualization focuses on a specific aspect and the views complement each other. Altogether, they can be used to gain some insights into causes of deviations. The different visualizations are explained in Section 8.3.1 to Section 8.3.3. In Section 8.3.4, we show that data mining techniques can also be used to extract useful knowledge about deviations from alignments. In each section, we show some screenshots on how the techniques are implemented in ProM.

For convenience, in the remainder of this chapter we use a basic oracle function orc. Given the net in Figure 8.1 and a trace σ_x in the log of the same figure with ID x $(1 \leq x \leq 8, x \in I\!N)$, $orc(\sigma_x)(\gamma_x) = 1$ (γ_1 to γ_8 are shown in Figure 8.2). Furthermore,

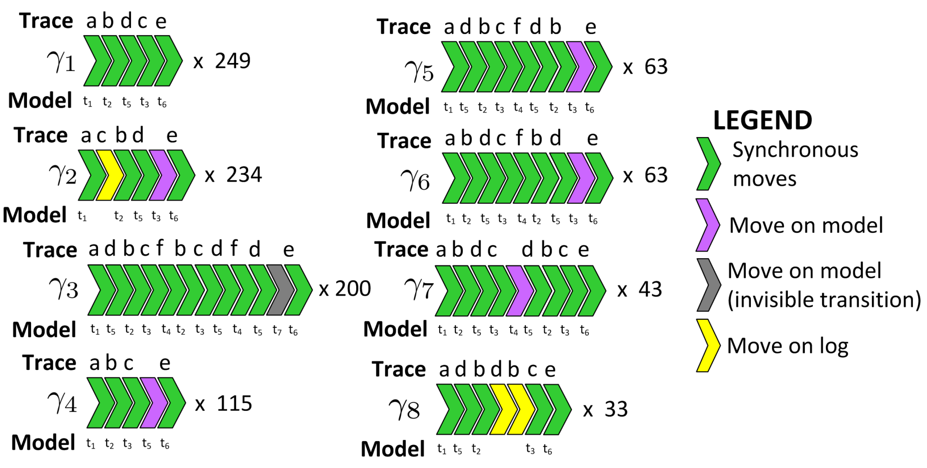


Figure 8.3: A “projection onto log” visualization of the alignments in Figure 8.2.

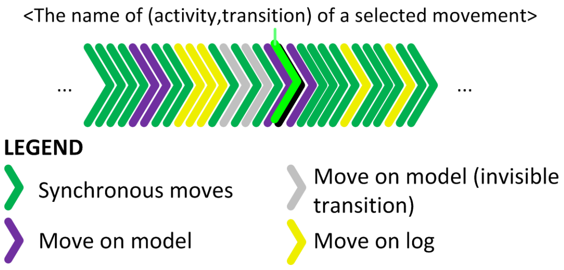

<The name of (activity,transition) of a selected movement>
...
...
LEGEND
Synchronous moves
Move on model
Move on model (invisible transition)
Move on log

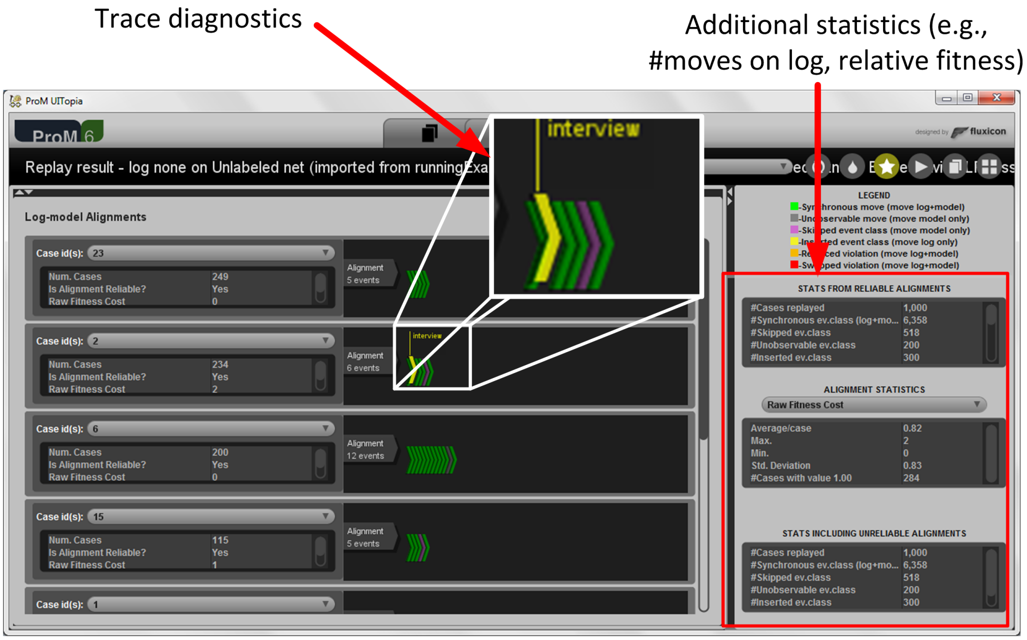
Figure 8.4: A compact visualization of the “projection onto log” where the width of arrows are narrow.

we define a multi-set of alignments Υ constructed from the net and all traces in the log of Figure 8.1 using orc, such that $\Upsilon = [\gamma_1^{249}, \gamma_2^{234}, \gamma_3^{200}, \gamma_4^{115}, \gamma_5^{63}, \gamma_6^{63}, \gamma_7^{43}, \gamma_8^{33}]$. In this example, there is a one-to-one correspondence between traces in the log and alignments in Υ. However, in principle there could be multiple alignments corresponding to a trace.

8.3.1 Projecting Alignments onto Logs

Given a multi-set of alignments, the first visualization shows each alignment independently as a sequence of “arrows” where each arrow is colored according to the type of movement it represents. We call this visualization a “projection onto log” as such a sequence resembles traces (i.e., sequence of activities) in the log. Green, yellow, purple, and grey colored arrows indicate *synchronous moves*, *moves on log*, *moves on model (visible transitions)*, and *moves on model (invisible transitions)* respectively. This visualization is intended to highlight *detailed deviations that occur in a trace* and *deviations that occur consecutively* at a glance.

Figure 8.3 shows all optimal alignments in Figure 8.2, projected onto the log of Figure 8.1. With this visualization, it is relatively easy to see which alignments have more deviations (i.e., moves on log/moves on model) than others and where deviations occur on each of the alignment. Notice that the visualization of γ_5 is similar to the visualization


Trace diagnostics
Additional statistics (e.g., #moves on log, relative fitness)
ProM UITopia
ProM 6
designed by fluxicon
Replay result - log none on Unlabeled net (imported from runningExa
interview
LEGEND
-Synchronous move (move log+model)
-Unobservable move (move model only)
-Skipped event class (move model only)
-Inserted event class (move log only)
-Replaced violation (move log+model)
-Swapped violation (move log+model)

Log-model Alignments

Case id(s): 23

Num. Cases	249
Is Alignment Reliable?	Yes
Raw Fitness Cost	0

Alignment 5 events

Case id(s): 2

Num. Cases	234
Is Alignment Reliable?	Yes
Raw Fitness Cost	2

Alignment 6 events

Case id(s): 6

Num. Cases	200
Is Alignment Reliable?	Yes
Raw Fitness Cost	0

Alignment 12 events

Case id(s): 15

Num. Cases	115
Is Alignment Reliable?	Yes
Raw Fitness Cost	1

Alignment 5 events

Case id(s): 1

STATS FROM RELIABLE ALIGNMENTS

#Cases replayed	1,000
#Synchronous ev.class (log+mo...	6,358
#Skipped ev.class	518
#Unobservable ev.class	200
#Inserted ev.class	300

ALIGNMENT STATISTICS

Raw Fitness Cost

Average/case	0.82
Max.	2
Min.	0
Std. Deviation	0.83
#Cases with value 1.00	284

STATS INCLUDING UNRELIABLE ALIGNMENTS

#Cases replayed	1,000
#Synchronous ev.class (log+mo...	6,358
#Skipped ev.class	518
#Unobservable ev.class	200
#Inserted ev.class	300

Figure 8.5: A screenshot of ProM, showing the multi-set of optimal alignments Υ, projected onto the log of Figure 8.1.

of γ_6 although they have a different sequence of movements.

The name of an activity in a log and the name of a transition in a Petri net can be lengthy. Annotating the names of all long activities/transitions on top/bottom of each arrow may yield unreadable visualization. Therefore, we make a "compact" version of the visualization. We narrowed the width of arrows and only show the name of activities/transitions of a movement when the movement is explicitly "selected". An illustration of such a visualization is shown in Figure 8.4. Not more than one movement can be selected at a time. This way, we ensure that the name of activities/transitions of a selected movement is always readable. Figure 8.5 shows a screenshot of the multi-set of alignments Υ, projected onto the log of Figure 8.1 as implemented in ProM. In the figure, move on log $(\mathit{interview}, \gg)$ in alignment γ_2 is highlighted.

8.3.2 Projecting Alignments onto Models

Given a process model (in the form of a Petri net) and a multi-set of alignments, the second visualization is a projection of all alignments in the multi-set onto the model. Unlike the projection onto log that shows deviations on each alignment in isolation, a projection of all alignments onto the model shows aggregated information about deviations. Thus, *this visualization shows insights into what type of deviations often occur and where they occur*.

Figure 8.6 shows a projection of all alignments in Figure 8.2 onto the Petri net shown in Figure 8.1. We use the color, size, and other decorations of Petri net elements (i.e., transitions, places, and arcs) to annotate the original net with information about the way the process was conducted and which deviations occurred. The color of a transition

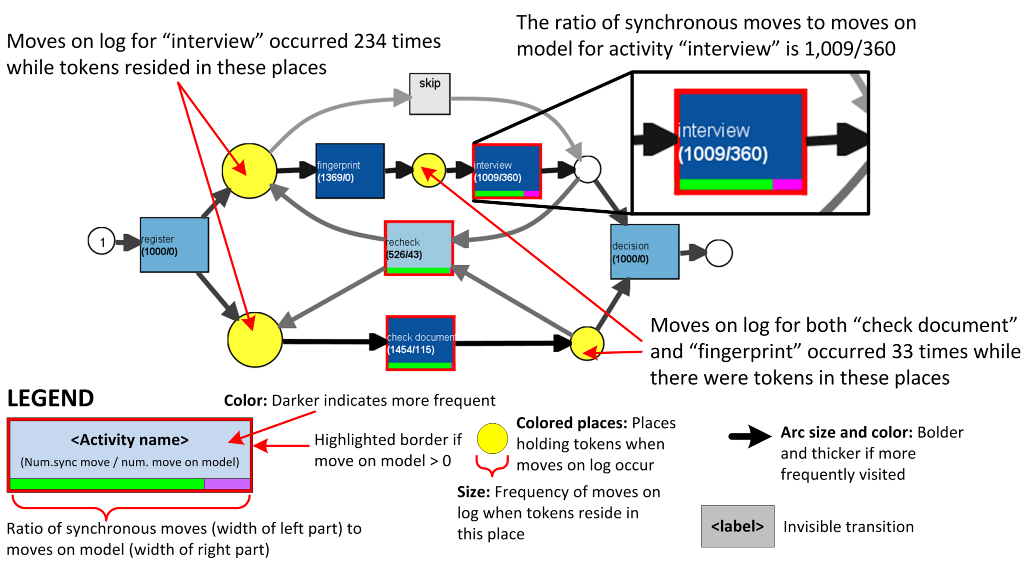
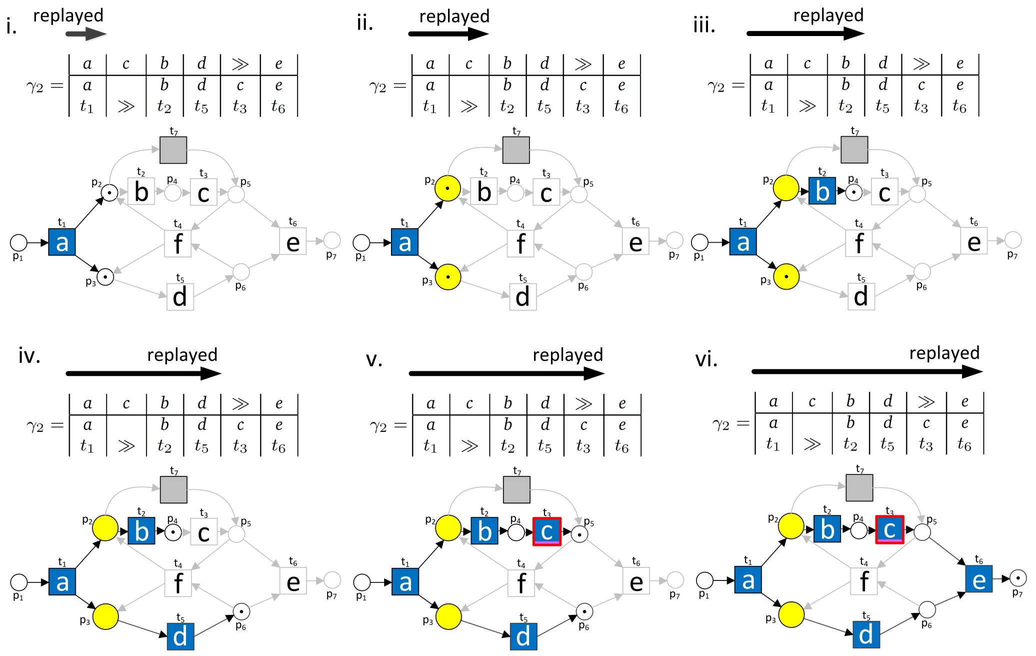

Moves on log for "interview" occurred 234 times while tokens resided in these places
The ratio of synchronous moves to moves on model for activity "interview" is 1,009/360
skip
fingerprint (1369/0)
interview (1009/360)
1
register (1000/0)
recheck (526/43)
decision (1000/0)
check document (1454/115)
Moves on log for both "check document" and "fingerprint" occurred 33 times while there were tokens in these places
LEGEND
Color: Darker indicates more frequent
<Activity name>
(Num.sync move / num. move on model)
Highlighted border if move on model > 0
Ratio of synchronous moves (width of left part) to moves on model (width of right part)
Colored places: Places holding tokens when moves on log occur
Size: Frequency of moves on log when tokens reside in this place
Arc size and color: Bolder and thicker if more frequently visited
<label> Invisible transition

Figure 8.6: A "projection onto model" of multi-set of alignments Υ.



a	c	b	d	$\gg$	e
a		b	d	c	e
t_1	$\gg$	t_2	t_5	t_3	t_6

Figure 8.7: Step-by-step on projecting alignment γ_2 Figure 8.2 onto the net in Figure 8.1.

in a projected Petri net indicates the frequency of synchronous moves. The darker it is in comparison to other transitions, the more it is visited. Similarly, the width and color of arcs also indicate the frequency of tokens that flows through them. The ratio of synchronous moves to moves on model for a transition is shown by the green and the purple bar below the transition. Note that a transition in the net has no such a bar if there is no move on model that involves the transition.

From the visualization in Figure 8.6, we know that three transitions were skipped (i.e., there are three transitions with some moves on model) while they should have been executed according to the net: interview, recheck, and check document (see the highlighted transitions). At a glance, we can clearly see that the total number of synchronous moves for the transition labeled interview is higher than the number of moves on model for the same transition (1,009 compares to 360).

Given a Petri net and a movement that changes the state of the net (either a synchronous move or a move on model), we know which transition in the net is correlated to the movement. Therefore, all information about either synchronous moves or moves on model in the alignment are projected onto the transitions of the net. However, a move on log does not contain any information about the elements of Petri nets. A Petri net may have multiple transitions with the same activity label. In such cases, projecting information about moves on log to transitions may yield misleading insights into deviations.

Therefore, moves on log are projected onto places of Petri nets. The marking of a Petri net shows explicitly the state of the net. Given an alignment of a Petri net, we "replay" the alignment on the net and increase the size of places with some tokens just before a move on log occurs. Take for example alignment γ_2 in Figure 8.2 and the net in Figure 8.1. Figure 8.7 illustrates the steps of projecting alignment γ_2 onto the net. The places of the markings reached just before some moves on log occur are colored yellow.

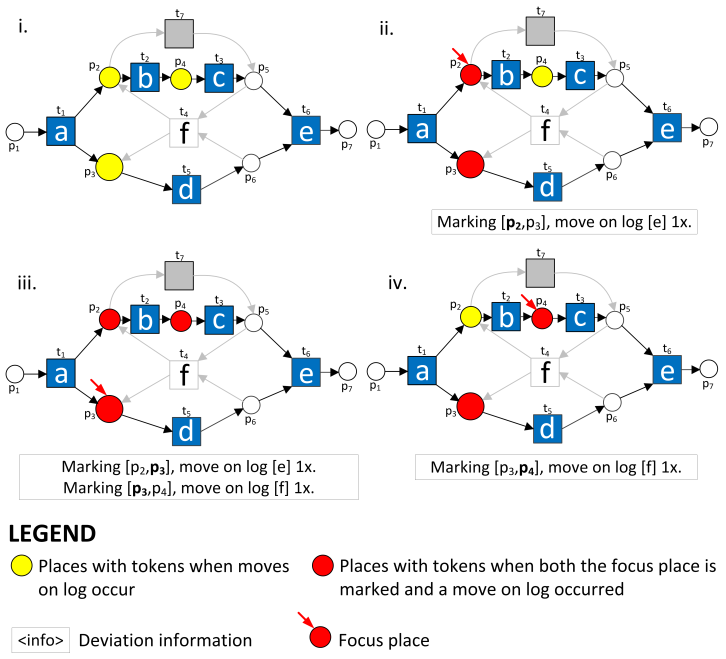
We introduce the notion of *focus places* to show the details on which markings are reached when some moves on log occur. Take for example alignment γ_9 in Figure 8.8 and its projection onto the net in Figure 8.1 (see Figure 8.8(i)). While replaying γ_9 on the net, two reachable markings are followed by some moves on log : (1) marking $[p_2, p_3]$ (reachable after replaying $\langle (a, t_1) \rangle$) is followed by move on log $(e, \gg)$, and (2) marking $[p_3, p_4]$ (reachable after replaying $\langle (a, t_1), (e, \gg), (b, t_2) \rangle$) is followed by move on log $(f, \gg)$. In this example, place p_3 is involved in both markings. Thus, the size of p_3 is larger than both p_2 and p_4. Given a focus place of the net, when replaying γ_9, we mark all places of the markings that both contain the focus place and are followed by a move on log. We annotate the net with the information on the places and frequencies of such moves on log, as shown in Figure 8.8.

Figure 8.9 shows a screenshot of the multi-set of optimal alignments Υ projected onto the net of Figure 8.1, implemented in ProM. The information panel in the figure shows that when the red-colored places are marked, moves on log for activity interview occurred 234 times in 234 traces. Note that this is the same number as the frequency of traces with id 2 in Figure 8.1.

8.3.3 Aligning Alignments

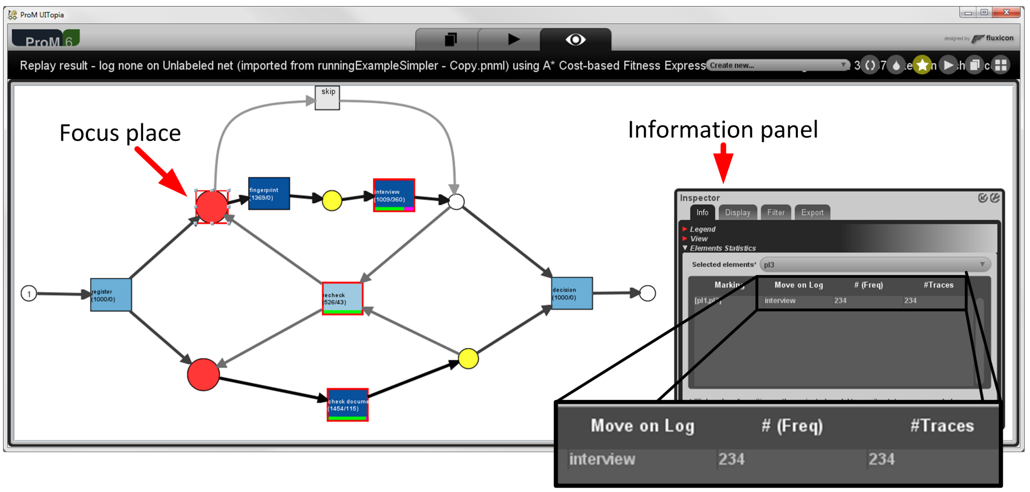
Given a multi-set of alignments derived from an event log and a Petri net, a projection onto the log of this multi-set shows detailed information about deviations per alignment. However, it is hard to aggregate deviation information from the visualization. In contrast, a projection onto model of the multi-set shows an aggregated view of what

$$\gamma_9 = \begin{array}{|c|c|c|c|c|c|c|}\hline a & e & b & f & c & d & e \\ \hline a & & b & & c & d & e \\ t_1 & \gg & t_2 & \gg & t_3 & t_5 & t_6 \\ \end{array}$$



i.
ii.
iii.
iv.
Marking [p2,p3], move on log [e] 1x.
Marking [p2,p3], move on log [e] 1x.
Marking [p3,p4], move on log [f] 1x.
Marking [p3,p4], move on log [f] 1x.
LEGEND
Places with tokens when moves on log occur
Places with tokens when both the focus place is marked and a move on log occurred
<info> Deviation information
Focus place

Figure 8.8: Top: An optimal alignment between trace $\sigma_9 = \langle a, e, b, f, c, d, e\rangle$ and the net in Figure 8.1, **Middle to Bottom: (i.)** a "projection onto model" of γ_9 to the net without any focus place. It is not obvious which moves on log occurred at which markings of the net. **(ii., iii., and iv.)** The same visualization with focus place p_2, p_4, and p_3, respectively.



Replay result - log none on Unlabeled net (imported from runningExampleSimpler - Copy.pnml) using A* Cost-based Fitness Express

Move on Log	# (Freq)	#Traces
interview	234	234

Figure 8.9: A screenshot of ProM, showing the projection of multi-set of optimal alignments Υ onto the net of Figure 8.1. The highlighted red-colored place is the selected focus place. The information panel shows the number of moves on log that occurred while tokens resided in the focus place.

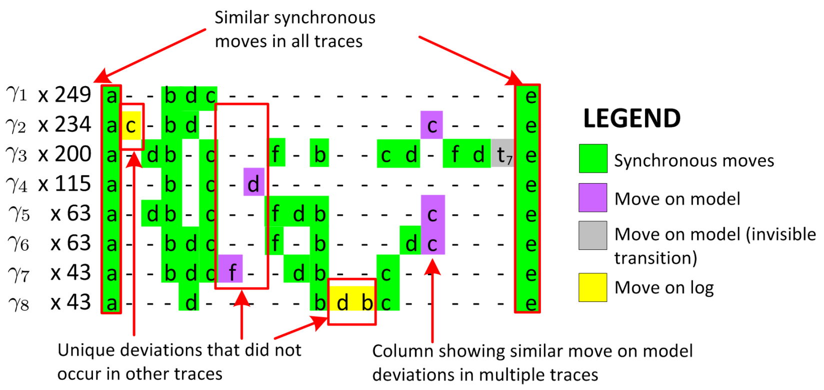
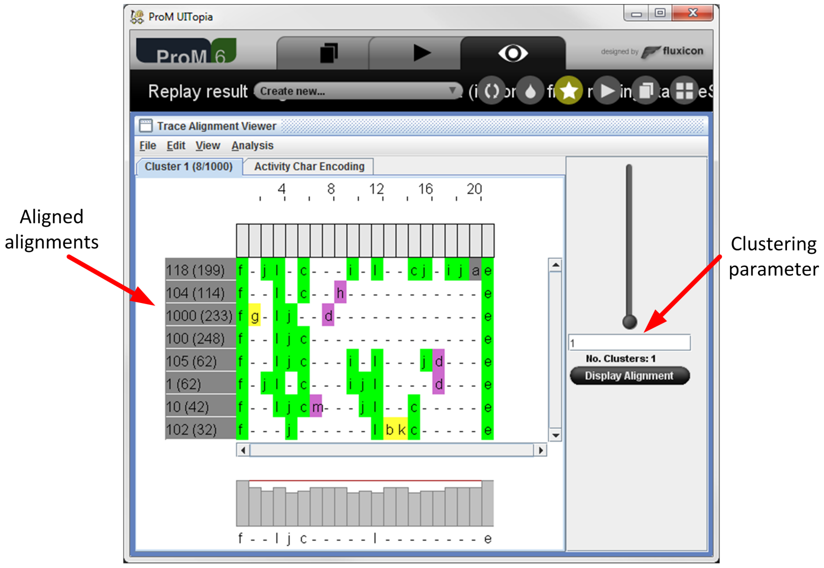

Similar synchronous moves in all traces
LEGEND
Synchronous moves
Move on model
Move on model (invisible transition)
Move on log
Unique deviations that did not occur in other traces
Column showing similar move on model deviations in multiple traces
γ1 x 249
γ2 x 234
γ3 x 200
γ4 x 115
γ5 x 63
γ6 x 63
γ7 x 43
γ8 x 43

Figure 8.10: An aligned multi-set of alignments Υ.

deviations often occur and where they occur, but the ordering of parallel activities is no longer shown. Inspired by the trace alignment technique in [21], the third visualization shows an aggregated view on deviations without losing completely the context in which deviations occur. The third visualization of the multi-set, namely the *aligned multi-set of alignments*, is a matrix that shows each alignment in a row and aligns the movements vertically between alignments such that each column in the matrix consists of only occurrences of a specific activity movement or gaps ("-"). Columns in the matrix are not allowed to contain only gaps or contain different movements. Moreover, the movements are aligned in a way that minimizes the number of gaps in all columns. Figure 8.10 shows an aligned alignments multi-set based on Figure 8.2.

A cell in an aligned multi-set of alignments is colored white if it contains a gap, otherwise it is colored according to the type of contained movements (i.e., green, yellow, purple, and grey for synchronous moves, moves on log, moves on model of non-invisible transitions, and moves on model of invisible transitions respectively). Figure 8.10 shows that all traces start with a synchronous move a and end with synchronous move e. It is easy to see from the visualization that some alignments have the same type of deviations in a certain "context", e.g., γ_2, γ_5, and γ_6 have a move on model just before a synchronous move of e. This visualization also shows deviations that only occur in some alignments, e.g., moves on log d and b on γ_8. Note that compared to the projection onto log of the same multi-set of alignments, the difference between γ_5 and γ_6 becomes clearer using this visualization (only the position of d is different). However, if the lengths of the alignments in the multi-set are long and each of them is unique then identifying the frequently occurring deviations with this visualization can be more difficult than identifying the same insights from the projection onto log of the same multi-set. Furthermore, it is hard to recognize parallel movements using this visualization.

Figure 8.11 shows the screenshot of aligned multi-set Υ, implemented in ProM. The implemented visualization also allows for clustering alignment based on the trace clustering technique in [21]. In cases where there are groups of "similar" alignments in a given multi-set of alignments, clustering them and aligning the alignments per cluster may yield better insights into deviations. Note that the visualization shown in Figure 8.11 is slightly different from the one shown in Figure 8.10 as the heuristics used to compute the aligned alignments do not guarantee that the aligned alignments has the least number of columns and rows.



118 (199)
104 (114)
1000 (233)
100 (248)
105 (62)
1 (62)
10 (42)
102 (32)

Figure 8.11: A screenshot of ProM, showing the aligned multi-set of alignments Υ using the approach in [21].

Table 8.1: Table representation of the multi-set of alignments shown in Figure 8.2 to mine frequently occurring deviations

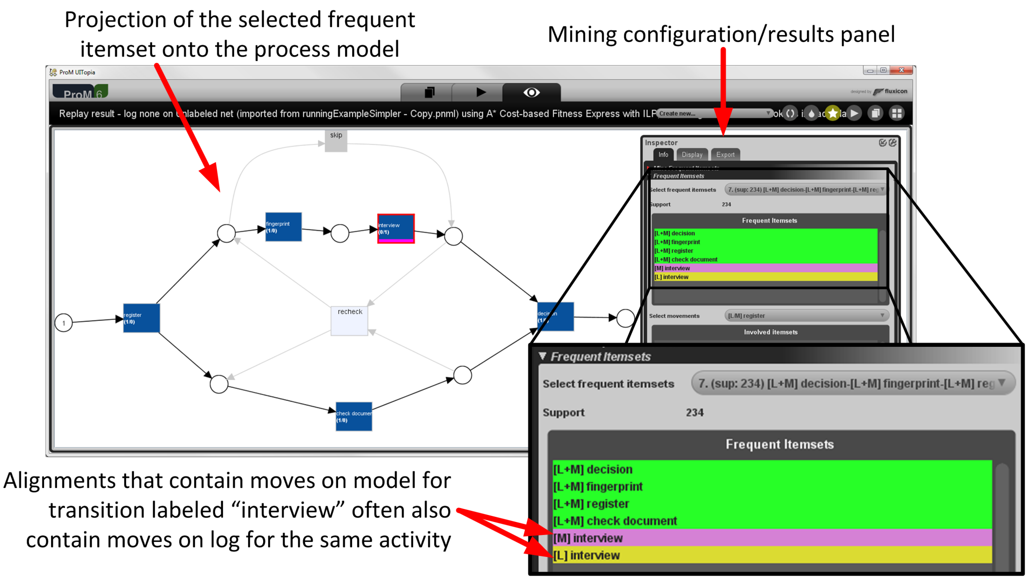
Alignment	**mvSync** a	**mvSync** b	**mvSync** c	...	**mvModel** c	...	**mvLog** c	...	**Frequency**
γ_1	1	1	1	...	0	...	0	...	249
γ_2	1	1	0	...	1	...	1	...	234
γ_3	1	2	2	...	0	...	0	...	200
γ_4	1	1	1	...	0	...	0	...	115
γ_5	1	2	1	...	1	...	0	...	63
γ_6	1	2	1	...	1	...	0	...	63
γ_7	1	2	2	...	0	...	0	...	43
γ_8	1	1	1	...	0	...	0	...	33

Deviation analysis using visual analytics relies on a human's perceptive ability to interpret interesting insights. In some cases, obtaining such insights may not be an easy task. Thus, in Section 8.3.4 we present some data mining approaches that can be used to complement the visual feedback provided in this section in order to obtain further insights into deviations from a given multi-set of alignments.

8.3.4 Data Mining Approaches to Analyze Deviations

Data mining is a discipline that covers all techniques to extract knowledge from large amounts of data [78]. Typically, the extracted knowledge is much less than the amount of data required to obtain it. Obviously, deviation analysis based on event logs and models fits the definition of problems that can be solved using data mining techniques. Given an event log and a process model, the number of deviations of interest between the logged activities and the modeled behavior is typically much less than the number of logged activities.

There are various data mining techniques that can be applied to analyze deviations


Projection of the selected frequent itemset onto the process model
Mining configuration/results panel
Alignments that contain moves on model for transition labeled "interview" often also contain moves on log for the same activity
ProM 6
Replay result - log none on unlabeled net (imported from runningExampleSimpler - Copy.pnml) using A* Cost-based Fitness Express with ILP
skip
fingerprint
interview
recheck
register
decision
check document
Inspector
Info
Display
Export

Frequent Itemsets

Select frequent itemsets 7. (sup: 234) [L+M] decision-[L+M] fingerprint-[L+M] reg

Support 234

Frequent Itemsets

[L+M] decision
[L+M] fingerprint
[L+M] register
[L+M] check document
[M] interview
[L] interview

Figure 8.12: A screenshot of the alignment frequent itemset visualization, implemented in ProM. The visualization shows a frequent itemset $\{(decision, t_6), (fingerprint, t_2), (register, t_1), (check\ document, t_5), (\gg, t_3), (interview, \gg)\}$ obtained by applying the FPGrowth mining algorithm [79] to the data set of Table 8.1. Moves on model and synchronous moves in the itemset are projected onto the original Petri net with dark color.

based on multi-sets of alignments as deviations are explicitly shown in the multi-sets. For example, frequent pattern mining techniques can be used to extract frequently co-occurring deviations. Frequent patterns in a data set are collections of data values that appear together in the data set frequently. For example, given a set of transactional data of items that people bought in a supermarket as a data set, a frequent pattern of this data set is a set of items that are often bought together. To identify the causes of deviations, we often interested in frequently occurring deviations as well as other deviations that occur together frequently with these deviations. Given a multi-set of alignments, the problem of finding a set of frequently occurring deviations can be translated into the problem of finding frequent patterns from a data set.

Take for example the multi-set of alignments Υ. We consider each movement in the multi-set as an item, and thus an alignment is a multi-set of items. We represent each possible movement of alignment multi-sets as a column of a table (see Table 8.1). This table can be further processed as a data set by frequent pattern mining algorithms. The number of occurrences of each movement in each alignment in an alignment multi-set is counted as the value of the movement attribute for the alignment. For example, there are two synchronous moves of c in γ_3. Therefore, the value of "mvSync c" (i.e., synchronous moves of activity c) of γ_3 in Table 8.1 is 2. Note that the last column in the table indicates the frequency of the alignment. This way, we also weigh each alignment according to its occurrences in the alignment multi-set.

Figure 8.12 shows the screenshot of the *alignment frequent itemset visualization*, implemented in ProM. The screenshot shows the results of applying the FPGrowth frequent itemset mining algorithm [79] to the data set shown in Table 8.1 with a minimum support value of 0.23 (i.e., a pattern is considered as a frequent pattern if 23% items in the data set contains the pattern). We take the existing implementation of FPGrowth al-

Table 8.2: Table representation of alignment multi-set shown in Figure 8.2 to mine possible causes of skipping activity interview (move model c)

Alignment	mvSync a	mvSync b	mvSync c	...	Frequency	mvModel c (interview) occur?
γ_1	1	1	1	...	249	No
γ_2	1	1	0	...	234	No
γ_3	1	2	2	...	200	No
γ_4	1	1	1	...	115	No
γ_5	1	2	1	...	63	Yes
γ_6	1	2	1	...	63	Yes
γ_7	1	2	2	...	43	No
γ_8	1	1	1	...	33	No

gorithm from the WEKA library.[1] Several frequent itemsets are found in the data set. One of the interesting itemsets consists of the following movements: $\{(decision, t_6), (fingerprint, t_2), (register, t_1), (check\ document, t_5), (\gg, t_3), (interview, \gg)\}$ (as shown in Figure 8.12). The projection of the itemset to the net of Figure 8.1 is shown in Figure 8.12. The transitions and their input/output arcs that correspond to some elements in the itemset are colored darker. The itemset shows that moves on model for transition c (i.e., interview) frequently occur together with moves on log for activity interview. Note that other frequent itemset mining algorithms, e.g., the apriori algorithm [14], can also be used to derive the same information.

Table representations of multi-sets of alignments such as the one shown in Table 8.1 can also be modified to allow for standard classification technique from data mining. Given a data set and a chosen attribute of the data set as a label, a classification technique builds a *classifier* to predict the value of the label attribute from the values of non-label attributes. We can use this type of technique to investigate the reasons why some deviations occur in a multi-set of alignments. A classifier that predicts the value of a label attribute accurately in a data set implies that it captures the knowledge required to determine the value of the attribute from other attributes. If the label attribute corresponds to a deviation, classifiers with an explicit representation such as decision trees may explain why a particular deviation occurs.

Given a multi-set of alignments and a particular deviation of interest, we construct a table representation of the multi-set in a similar way as the example shown in Table 8.1. In addition, we add an extra label attribute (column) identifying the deviation of interest. Then, we use existing classification techniques to build a classifier for the added label attribute.

Take for example the multi-set of alignments Υ. Suppose that we are interested to know the possible causes of moves on model for transition with label c, i.e., an interview needed according to the model is skipped in reality. We construct a table representation of the multi-set as shown in Figure 8.2 and add an extra attribute that indicates whether moves on model c occurred in each alignment. Note that Table 8.2 has an extra label attribute (i.e., extra column) and has no "mvModel c" column in comparison with Table 8.1 because the latter column is used to derive the new label attribute. We use existing classification algorithms that use an explicit classifier such as decision trees to explain under which circumstances "mvModel c" column has value of "Yes" or "No".

We implemented a plugin in ProM to export a multi-set of alignments to a CSV format

[1]see `http://www.cs.waikato.ac.nz/ml/weka/`

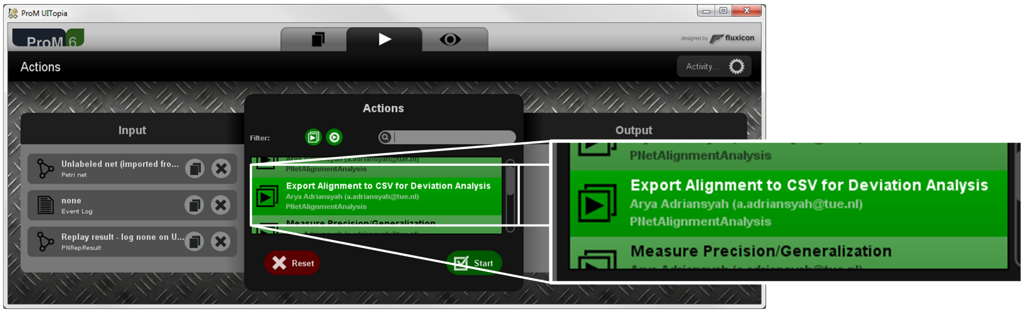

ProM UITopia
ProM 6
Actions
Activity...
Input
Actions
Output
Filter:
Unlabeled net (imported fro...
Petri net
none
Event Log
Replay result - log none on U...
PNRepResult
Export Alignment to CSV for Deviation Analysis
Arya Adriansyah (a.adriansyah@tue.nl)
PNetAlignmentAnalysis
Measure Precision/Generalization
Reset
Start

Figure 8.13: Screenshot of the ProM plugin that is used to export a multi-set of alignments to a CSV file in a format similar to Table 8.1.

file similar to the one shown in Table 8.1. The screenshot of the plug-in is shown in Figure 8.13. Furthermore, instead of having a special column that shows the frequency of each unique alignment in the multi-set, we list all of the alignments in the CSV file. This way, we can also insert the attributes of traces where the alignments are derived from whenever it is necessary. This CSV file can be used as an input for many existing data mining tools.

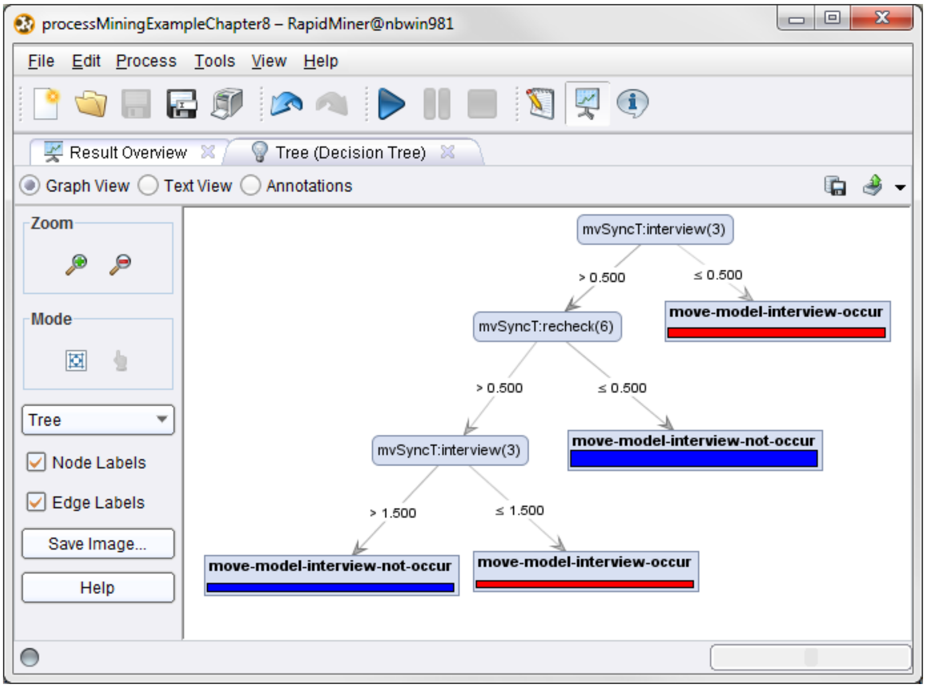


Figure 8.14: A screenshot of Rapidminer data mining tool, showing a decision tree mined from multi-set of alignments Υ to predict the occurrences of moves on model for transition c (interview) using the C4.5 algorithm [135].

Figure 8.14 shows a screenshot of the Rapidminer[2] data mining tool, used to mine

[2]see `http://rapid-i.com`

a decision tree that predict the occurrences of move on model for transition labeled c (interview) in alignment multi-set Υ. For each node in the tree, mvSyncT:X is a label of a column that stores the number of occurrences of synchronous moves for activity X. The tree shows that an alignment in the multi-set has a move on model for the transition labeled c if (1) there is no synchronous move for c (i.e., the value of mvSyncT:interview ≤ 0.5), or (2) there is a synchronous move for f (recheck) and a synchronous move for c (interview) occurred exactly once (i.e., the value of mvSyncT:recheck > 0.5 and $0.5 <$ the value of mvSyncT:interview ≤ 1.5). Obviously, C4.5 is not the only algorithm that can be used to extract such insights into deviations. Other decision-tree classification algorithms such as ID3 [134], Bayesian network classification algorithms (e.g., [58, 110]), and rule-based classification algorithms (e.g., [84]) can also be used.

Other than the frequent set mining and classification techniques, there are many other types of data mining techniques that can be used to analyze recurring deviations given a table representation of an alignment multi-set. For example, clustering techniques can be used to create clusters of alignments, where each cluster indicates the group of traces that have similar deviations. As another example, data mining outlier detection techniques can be used to identify exceptional type of deviations.

8.4 Experiments

To show the applicability of the approach, we performed two case studies using two pairs of real-life logs and models. These illustrate that our diagnostics provide insights into causes of deviations in real-life settings. In the first case study (see Section 8.4.1), we show that the approach shows insights into possible root causes of deviations between the observed behavior in a log and a reference model. In the second case study (see Section 8.4.2), we show that the obtained insights from the approach can even be exploited to complement process discovery techniques. For the data mining-related analyses used in the experiments, we use standard data mining tools like WEKA and Rapidminer.

8.4.1 Dutch Municipality Case Study

In this first case study, we took an event log and a reference process model of handling building permits in a Dutch municipality. The reference model of this process is shown in Figure 8.15. Although the process is supported by a process-aware information system, its executions were not strictly enforced to follow the model. Thus, deviations to the model may exist in the log.

The process starts when a building permit request is accepted (Ontvangst aanvraag). After an acknowledgement is sent (Verzenden ontvangst bevestiging), the process splits into three parallel branches. The top branch starts with two activities that can be performed in any order, followed by a long sequence of activities with a possible skip on performing admissibility test (i.e., Ontvankelijkheidstoets). Similarly, the middle branch also consists of a sequence of activities with some allowed skips in between. The bottom branch consists of a sequence of activities required to make a decision about the request. None of these activities can be skipped. These activities are: (1) estimating the date of decision (Besluit datum besluit), (2) sending the decision date (Besluit datum verzenden), and (3) publishing the decision date (Besluit datum publicatie).

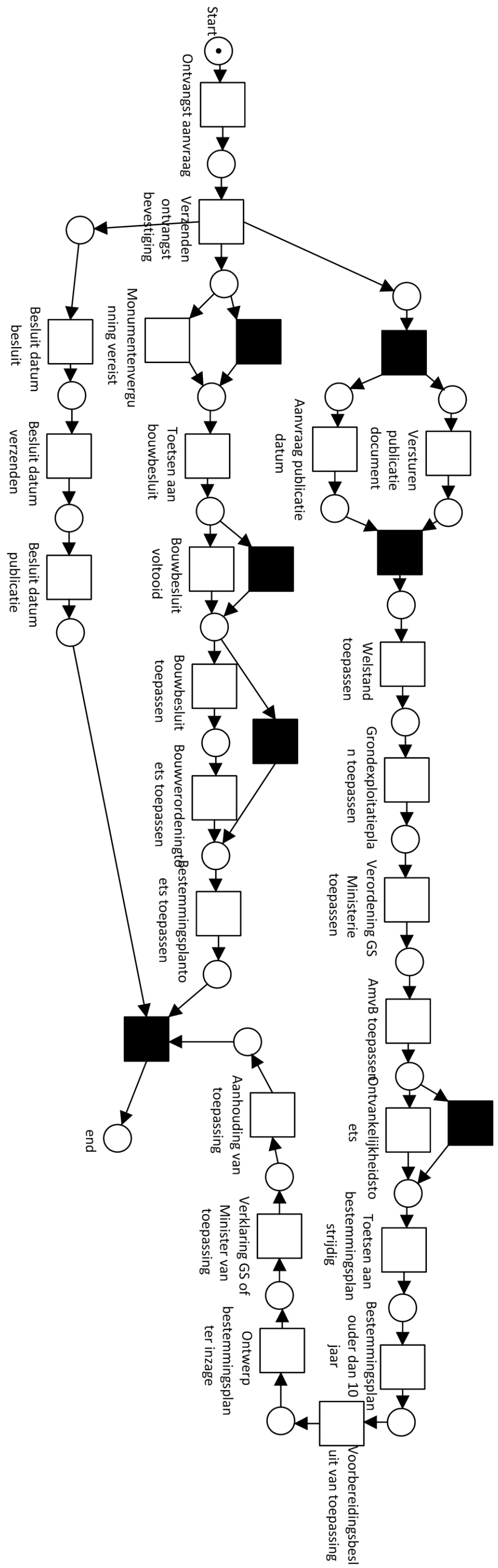

Start
Ontvangst aanvraag
Verzenden ontvangst bevestiging
Monumentenvergunning vereist
Besluit datum besluit
Besluit datum verzenden
Besluit datum publicatie
Toetsen aan bouwbesluit
Bouwbesluit voltoid
Bouwbesluit toepassen
Bouwverordeningets toepassen
Bestemmingsplantoets toepassen
Aanvraag publicatie datum
Versturen publicatie document
Welstand toepassen
Grondexploitatieplan toepassen
Verordening GS Ministerie toepassen
AmvB toepassen
Ontvankelijkheidstoets
Toetsen aan bestemmingsplan strijdig
Bestemmingsplan ouder dan 10 jaar
Voorbereidingsbesluit van toepassing
Ontwerp bestemmingsplan ter inzage
Verklaring GS of Minister van toepassing
Aanhouding van toepassing
end

Figure 8.15: A reference model for building permit application handling process.

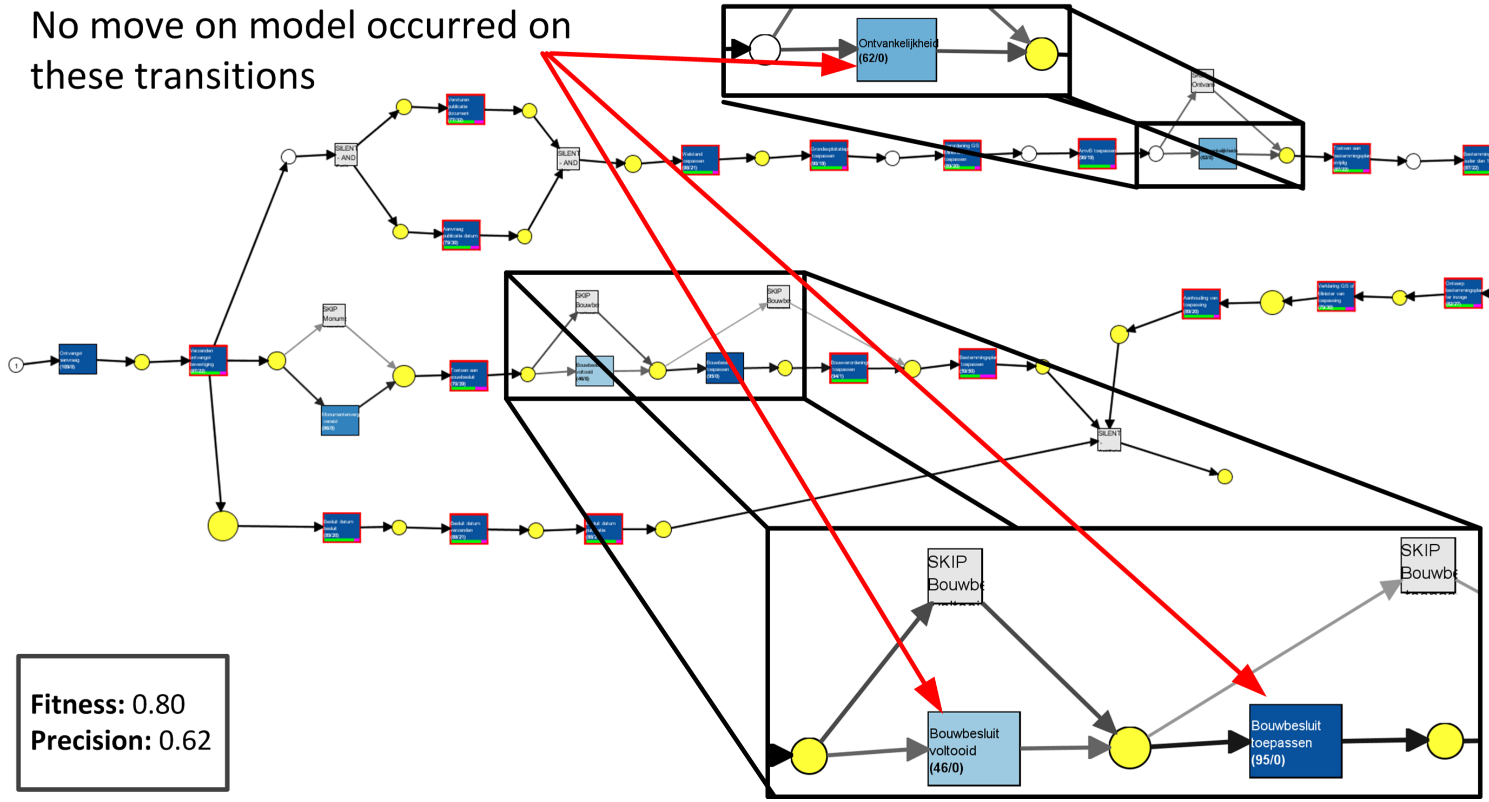

No move on model occurred on these transitions
Ontvankelijkheid (62/0)
SKIP Bouwbe
Bouwbesluit voltooid (46/0)
Bouwbesluit toepassen (95/0)
Fitness: 0.80
Precision: 0.62

Figure 8.16: Projection of diagnostics onto the model using the multi-set of optimal alignments constructed from a Dutch municipality log and the model shown in Figure 8.15. There is one-to-one mapping between each trace in the log with an alignment in the multi-set. No move on model ever occurred to the three highlighted visible transitions because they can be skipped.

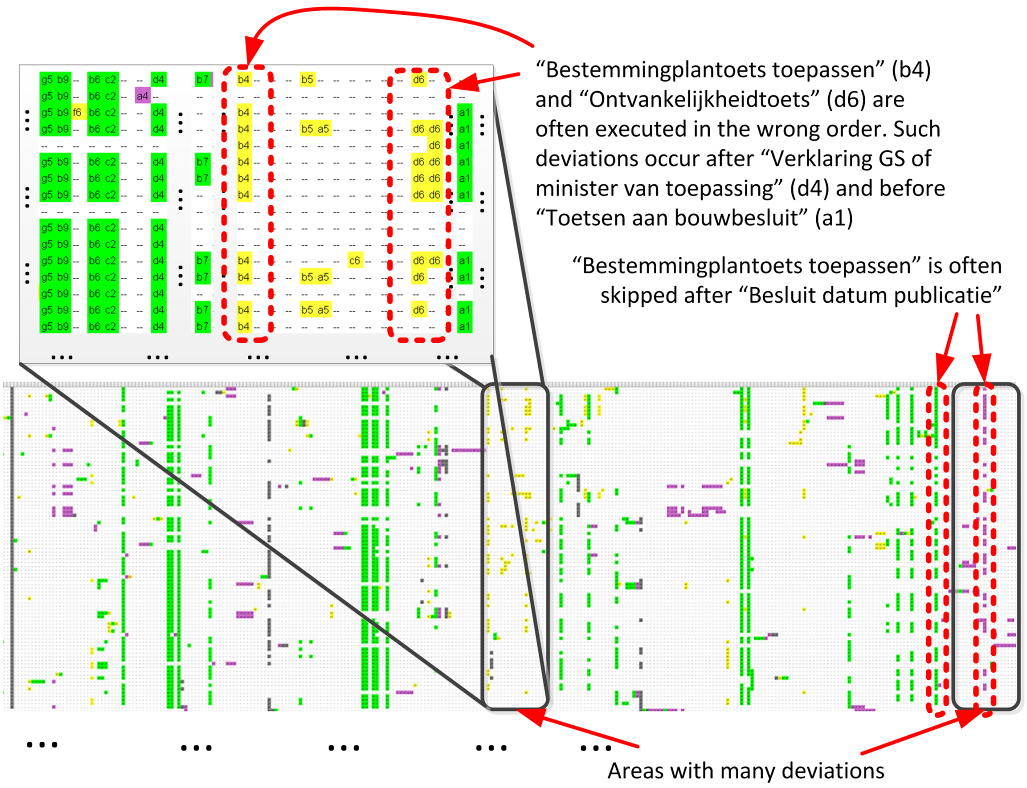

"Bestemmingplantoets toepassen" (b4) and "Ontvankelijkheidtoets" (d6) are often executed in the wrong order. Such deviations occur after "Verklaring GS of minister van toepassing" (d4) and before "Toetsen aan bouwbesluit" (a1)
"Bestemmingplantoets toepassen" is often skipped after "Besluit datum publicatie"
Areas with many deviations

Figure 8.17: An aligned multi-set of optimal alignments of a Dutch municipality log and its model (Figure 8.16).

To gain insights into possible deviations, we constructed a multi-set of alignments from the log and the model using a basic and optimal oracle function with respect to the standard cost function. We also measured the relative fitness and precision (using one optimal alignment per trace, both backward/forward constructed multi-set automata) between the log and the model. Figure 8.16 shows a projection onto model of the multi-set. As shown in the figure, many deviations occur in the log. Almost all places in the figure are colored, which indicates that many activities were performed when they were not allowed to according to the model. Similarly, almost all transitions have been skipped in some cases as shown by the number of highlighted transitions with small bars at their bottom. For a reference model that drives process enactments supported by an information system, a relative fitness value of 0.80 indicates that deviations frequently occurred (i.e., roughly 1 out of 5 activities were deviating). The low precision value (0.62) for the model also indicates that the model allows for much more behavior than the observed behavior in the log.

Figure 8.17 shows an alignment of the constructed multi-set of alignments. From this visualization, we identify at least two areas for which many deviations are observed. The first area refers to events between synchronous moves for Verklaring GS of minister van toepassing and synchronous moves for Toetsen aan bouwbesluit. Activity Bestemming toepassen followed by Ontvankelijkheidtoets were performed frequently between the two synchronous moves. Another area where deviations frequently occur is located just after the last synchronous move of Besluit datum publicatie. The large number of unique traces

Table 8.3: Conformance measurements of logs derived from the Dutch municipality log with respect to the original process model.

	Log	#Traces	#Events	Model	Relative Fitness	Precision
Original Log	Bouw4	109	2,331	Figure 8.16	0.80	0.62
Derived Log	L_1	67	1,442	Figure 8.16	0.74	0.60
	L_2	42	889	Figure 8.16	0.89	0.53
	L'_1	49	1,201	Figure 8.16	0.84	0.57

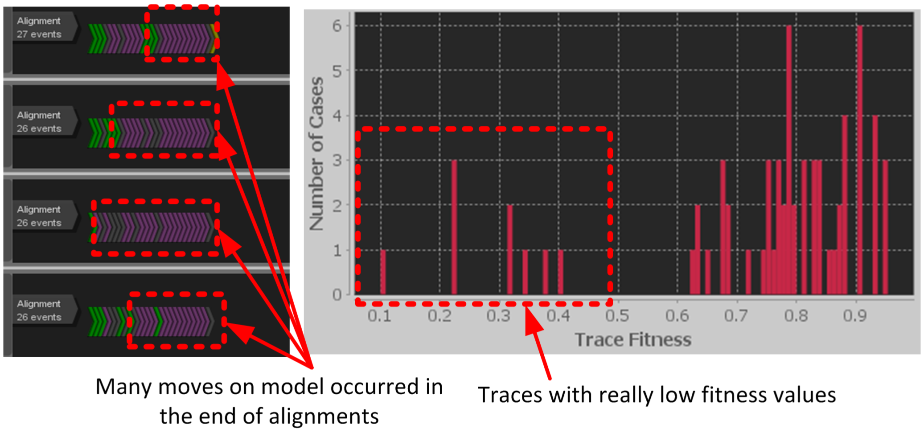
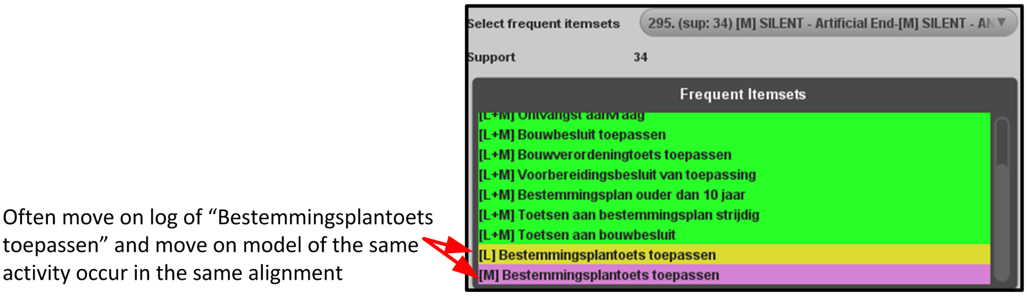


Figure 8.18: Left: Projection onto log of optimal alignments between some traces of L_1 and the net in Figure 8.16, and **Right:** A histogram showing the distribution of relative fitness of alignments between traces of L_1 and the net in Figure 8.16.

that deviate in those two areas implies that the reference model is not accurately describing a large part of the observed behavior. Thus, we split the log into two parts L_1 and L_2. L_1 contains all traces in the log whose alignments show deviations in some of the highlighted area. L_2 contains the rest of the traces of the original log. We measured the fitness and precision of each part with respect to the original net and show it in Table 8.3.

As shown in Table 8.3, L_1 has a lower relative fitness value than L_2 (0.74 compared to 0.89). L_1 has a slightly higher precision value than L_2 (0.60 compared to 0.53), but as discussed in Chapter 7, the precision value between a log and a model can be misleading if the fitness value is rather low. We investigated the causes of deviations of L_1 by projecting the multi-set of alignments for L_1 onto the log (see the left side of Figure 8.18). The figure shows that many alignments have a lot of moves on model towards the end of the process. This indicates that many of the traces in L_1 are incomplete. The right side of Figure 8.18 shows the distribution of relative fitness values of alignments between traces of L_1 and the model. The x-axis in the figure shows fitness values, while the y-axis shows the number of traces in L_1. As shown by the figure, some traces have very low fitness values compared to other traces.

To identify the root causes of deviations, we filtered out the incomplete traces in L_1. We noticed that some short incomplete traces have relatively high fitness values (around 0.6) because the ratios of the number of moves on model to the length of their optimal alignments are relatively low. Therefore, we kept only the traces whose optimal alignments have relative fitness value of 0.7 or higher and name the filtered log L'_1. We ran a frequent itemset mining algorithm (see Section 8.3.4) on the multi-set of alignments for


Often move on log of "Bestemmingsplantoets toepassen" and move on model of the same activity occur in the same alignment
Select frequent itemsets
295. (sup: 34) [M] SILENT - Artificial End-[M] SILENT - AN
Support
34
Frequent Itemsets

[L+M] Ontvangst aanvraag
[L+M] Bouwbesluit toepassen
[L+M] Bouwverordeningtoets toepassen
[L+M] Voorbereidingsbesluit van toepassing
[L+M] Bestemmingsplan ouder dan 10 jaar
[L+M] Toetsen aan bestemmingsplan strijdig
[L+M] Toetsen aan bouwbesluit
[L] Bestemmingsplantoets toepassen
[M] Bestemmingsplantoets toepassen

Figure 8.19: A frequently co-occurring movements identified from the multi-set of alignments for log L_1' and the model shown in Figure 8.16: moves on log for Bestemmingplantoets toepassen and move on model for Bestemmingplantoets toepassen often occur together (in 34 traces out of 49).

L_1' to identify the most frequently occurring deviations. The result of this analysis shows that 34 out of 49 alignments in the multi-set of L_1' have both a move on log for activity Bestemmingsplantoets toepassen and a move on model for the transition labeled with the same activity (see Figure 8.19). This indicates that the transition labeled Bestemmingsplantoets toepassen may occur at different times than what is described in the model.

The aligned multi-set of alignments in Figure 8.17 shows that the moves on log for activity Ontvankelijkheidtoets occurred more frequently than the moves on log for activity Bestemmingplantoets toepassen. Therefore, we performed an analysis using a classification algorithm technique to identify possible causes of move on log for activity Ontvankelijkheidtoets. Figure 8.20 shows a decision tree constructed by applying the C4.5 algorithm on the data set derived from the alignments of L_1' and model in Figure 8.16.

As shown by the decision tree, a move on log for activity Ontvankelijkheidtoets occurred if there is no synchronous move for the same activity (the frequency of mvSyncT:Ontvankelijkheidstoets is lower than or equal to 0.5). This is supported by 32 out of the 49 traces in the experiment. Interestingly, the decision tree also shows a possible relation between the occurrences of synchronous moves for activity Bestemmingplantoets toepassen and the occurrences of moves on log for activity Ontvankelijkheidtoets. Suppose that a synchronous move for activity Ontvankelijkheidtoets occurred, a move on log for activity Ontvankelijkheidtoets often co-exists with a synchronous move for activity Bestemmingplantoets toepassen. If a synchronous move for activity Bestemmingplantoets toepassen occurred then a move on log for activity Ontvankelijkheidtoets also occurred (supported by 6 traces). Similarly, if a synchronous move for activity Bestemmingplantoets toepassen did not occur then a move on log for activity Ontvankelijkheidtoets also did not occur (supported by 10 traces). This information can be very useful for process experts in order to understand the context where deviations often occur.

8.4.2 Dutch Financial Institution

In the second case study, we use the log provided for the BPI Challenge 2012 [192]. The log is taken from a Dutch financial institution and describes applications for personal loans or overdraft handling. The log contains the events recorded from three intertwined subprocesses: subprocess for processing applications (subprocess A), offers (O), and work items (W). The A subprocess is concerned with handling the loan applications received from customers. The O subprocess specifies how offers are send to customers. The W subprocess describes how work items, belonging to an application, are processed.

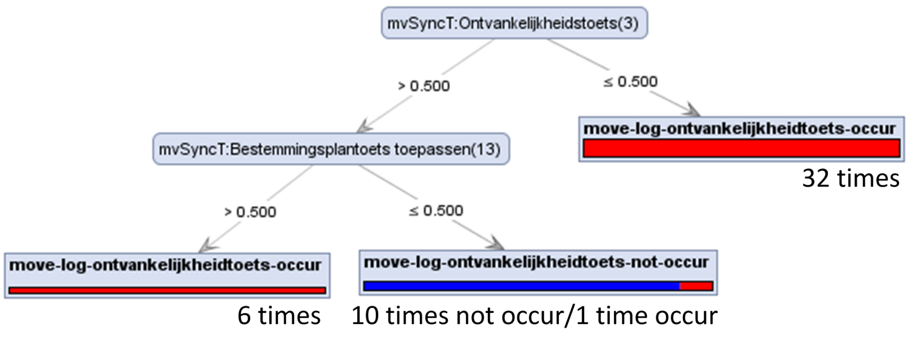

mvSyncT:Ontvankelijkheidstoets(3)
> 0.500
≤ 0.500
mvSyncT:Bestemmingsplantoets toepassen(13)
move-log-ontvankelijkheidtoets-occur
32 times
> 0.500
≤ 0.500
move-log-ontvankelijkheidtoets-occur
move-log-ontvankelijkheidtoets-not-occur
6 times
10 times not occur/1 time occur

Figure 8.20: A decision tree, showing the possible causes why moves on log for activity Ontvankelijkheidtoets occurred/not occurred.

In this case study, we only describe the analysis on the O subprocess. For a complete report on the case study, readers are referred to [3].

One of the interest of the process owner is to know what the process models actually look like. This is a typical question that often can be answered by applying process discovery algorithms to the log. However, later we show that this is not the case. Due to the limitations of many process discovery algorithms, it is important to first ensure that the traces in the log are completed. Thus, we filtered out unfinished cases in the log with the help of the dotted chart visualization [161]. The filtered log consists of 2,836 cases with total 17,572 events. We added an artificial start and end activities for all traces in the log to uniquely mark the start and end of each trace. Then, we use three different process discovery algorithms (i.e., the α-algorithm [1], heuristic miner [204], and ILP miner [190]) to construct three different process models of the filtered log.

Given the filtered log, the α-algorithm does not produce an easy sound model while both the ILP and the heuristic miner discover overly complex models. For each model, we constructed a multi-set of optimal alignments using a basic and optimal oracle function with respect to the standard likelihood cost function. Figure 8.21 and Figure 8.22 show a projection onto the model of the constructed multi-sets of alignments, discovered using the ILP miner and the heuristic miner respectively. The model discovered by the ILP miner (see Figure 8.21) allows the execution of activities O_DECLINED, O_ACCEPTED, and O_SENT_BACK as many times as possible when they are enabled. Moreover, the precision of the model is really low (0.46) although it has a perfect fitness value. Thus, it does not yield any useful insights into the way the process was performed. The model discovered using the Heuristic Miner (see Figure 8.22) has a lower fitness value than the model discovered by the ILP miner (0.75), but it has a higher precision value (0.79). However, Figure 8.22 shows that some transitions and arcs in the latter model are never used. The problems exist for both nets. This indicates that they are not sufficiently good enough to describe the way process was conducted.

In this section, we show that various visualizations explained in Section 8.3 can be exploited to *guide process model repair* to better reflect the reality. We followed the general steps in Figure 8.24, shown as a BPMN model to obtain a Petri net that describes the behavior of subprocess O in the log accurately. Given an event log, the first step shown in Figure 8.24 is to obtain an easy sound Petri net as an initial model. Such a

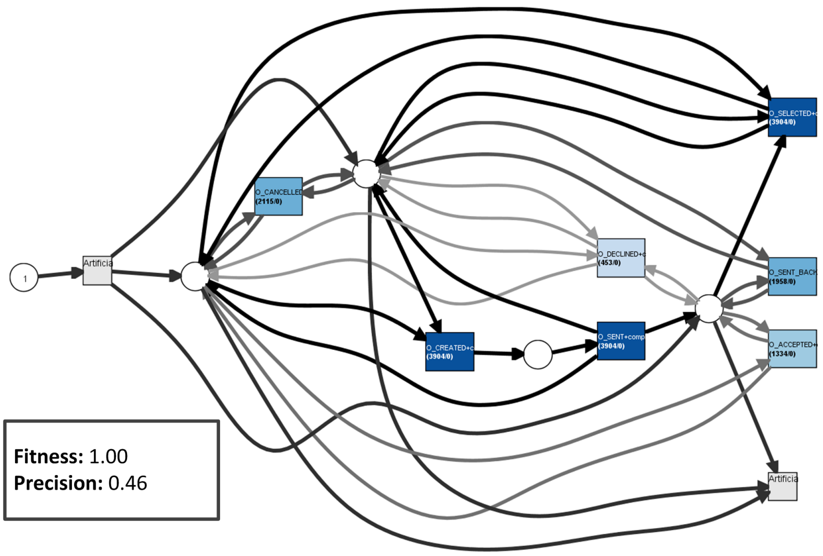


Figure 8.21: A projection onto model of an alignment multi-set of the filtered log for subprocess O with respect to a Petri net mined from the log using the ILP Miner [190].

net can be discovered from the log using existing process discovery algorithms or simply hand made according to some background knowledge about the process. Then, we constructed a multi-set of alignments in order to obtain some insights into causes of deviations. We use a basic and optimal oracle function with respect to the standard likelihood cost function to compute such a multi-set. We used the visualizations/data mining techniques presented in Section 8.3 to analyze the frequently occurring deviations. New transitions/places are then appended to the original net based on the analysis result *without restricting the behavior of the original net*, i.e., if a marking is reachable in the original net, it must also be reachable in the appended net. The two last steps are performed iteratively until a net with a sufficient *relative fitness* value is obtained. Last, we simplify the net by removing infrequent transitions and surrounding places to improve precision/generalization value. The "removal" of a transition in a Petri net that is either (1) very infrequently visited, or (2) can be reduced without changing the sequence of

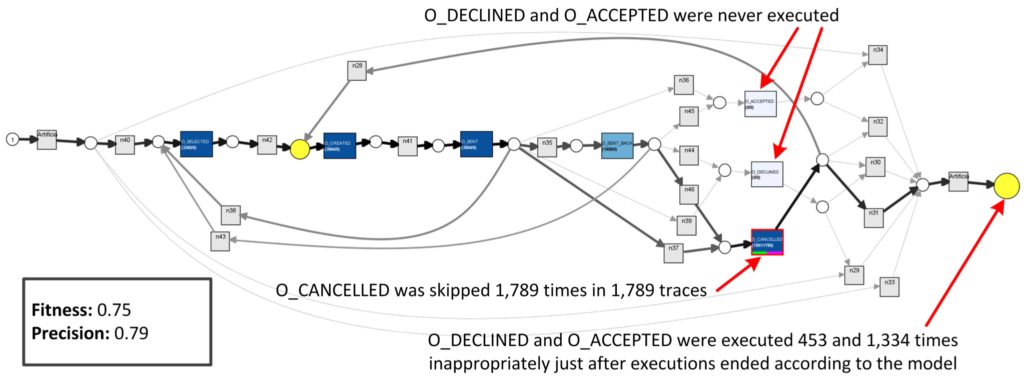

O_DECLINED and O_ACCEPTED were never executed
O_CANCELLED was skipped 1,789 times in 1,789 traces
O_DECLINED and O_ACCEPTED were executed 453 and 1,334 times inappropriately just after executions ended according to the model
Fitness: 0.75
Precision: 0.79

Figure 8.22: A projection onto model of an alignment multi-set of the filtered log for subprocess O with respect to a Petri net mined from the log using the Heuristic Miner [204].

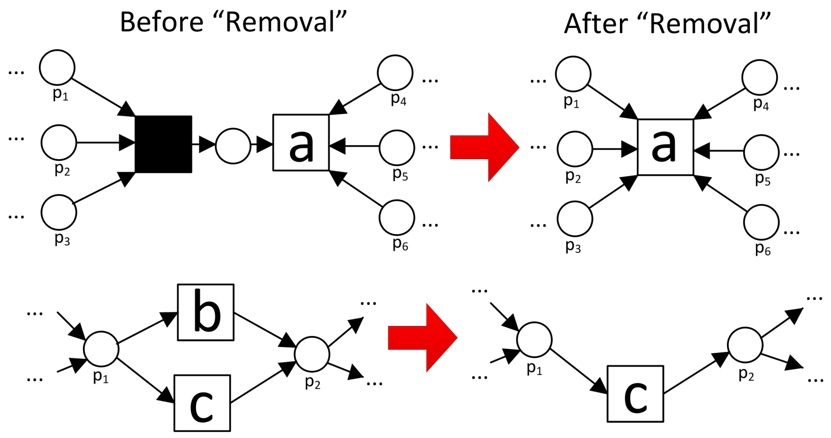


Figure 8.23: Two approaches to "remove" transitions: **Top:** removing invisible transitions using standard *reduction* techniques [119]. **Bottom:** removing infrequent transitions and their input/output arcs (here the transition with label b is removed).

activities allowed by the original net may only increase precision values. Such a reduced net has less number of transitions than the original one, thus it allows less behavior than the original net. We apply Petri net reduction techniques and simple transition removal to improve the conformance values increase in general. The proposed "removal" approaches are shown in Figure 8.23.

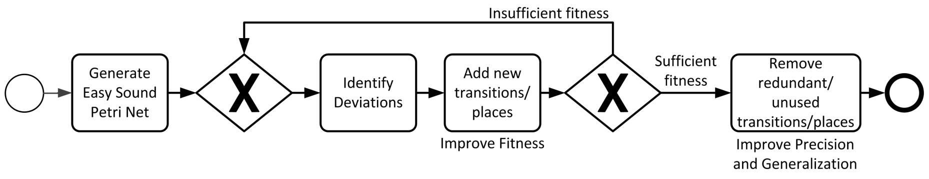

Insufficient fitness
Generate Easy Sound Petri Net
Identify Deviations
Add new transitions/ places
Improve Fitness
Sufficient fitness
Remove redundant/ unused transitions/places
Improve Precision and Generalization

Figure 8.24: General steps to repair process model to better reflect reality.

We used the net discovered by the heuristic miner (see Figure 8.22) instead of the one discovered by the ILP miner as an initial model because the fitness values of the net can still be improved by adding new transitions. As shown in Figure 8.22, activities O_DECLINED and O_ACCEPTED were frequently performed after the termination of the net had been reached (i.e., moves on log of both activities often occurred when there is a token in the colored end place). The figure also shows that the transition labeled O_CANCELLED was skipped many times. Interestingly, the frequency of moves on model on O_CANCELLED is exactly the same as the sum of frequency of move on log for both activities O_DECLINED and O_ACCEPTED. Thus, we add a choice to the input place of transition labeled O_CANCELLED such that both O_DECLINED and O_ACCEPTED can be performed when the transition is enabled. Projection on model of the multi-set of alignments between the traces of the log and the appended model is shown in Figure 8.25. Note that the previously appended transitions and arcs only allow more behavior and are not restricting the behavior of the original Petri net. The relative fitness value of the new alignment multi-set is higher than the original alignment multi-set (0.97

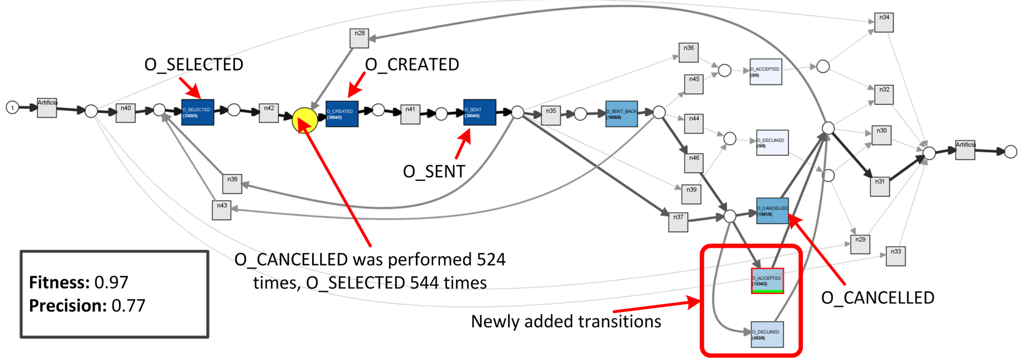


Figure 8.25: The model shown in Figure 8.22 after the first repair.

compared to 0.75), but its precision value is lower (0.79 compared to 0.77).

The colored places on the projection onto model in Figure 8.25 show that there is still some room for improvement by allowing extra behavior in the model. Thus, we perform a second iteration to repair the net in Figure 8.25. As shown by the figure, two moves on log for activities O_CANCELLED and O_SELECTED often occur at the same marking with approximately the same frequency. An aligned multi-set of alignments between the log and the net in Figure 8.25 shows that there is a repeated pattern of move on log occurrences in the alignment multi-set (see Figure 8.26). All moves on log occur just before a synchronous move for activity O_CREATED. Moreover, we identify two reoccurring sequences, each involves a move on log (see Figure 8.26). This implies that there is a loop in the alignment that ends with moves on log. Both loops start with synchronous moves O_CREATED followed by O_SENT, then followed by either a sequence of synchronous move O_SELECTED and a move on log O_CANCELLED or a sequence of synchronous move O_CANCELLED and a move on log O_SELECTED. Note that O_SELECTED and O_CANCELLED are swapped in both loops. Based on this analysis, we append the net to allow O_SELECTED and O_CANCELLED in any order just after activity O_SENT. After both activities are performed, O_CREATED has to be enabled such that it may loop. The appended net based on this analysis is shown in Figure 8.27. Note that the fitness value of the log with respect to the new appended net in Figure 8.27 is 0.99.

We performed similar steps as we performed before to repair the process model in Figure 8.27. An aligned multi-set of alignments between the log and model in Figure 8.27 is shown in Figure 8.28. As shown in the figure, we observed repeating blocks and regularity of moves on log for activity O_SENT_BACK. We appended the net to allow O_SENT_BACK between the occurrences of O_SENT and either O_SELECTED or O_CANCELLED. A projection onto model of a multi-set of alignments constructed from the newly appended net is shown in Figure 8.29.

The multi-set of alignments for the filtered log and the net shown in Figure 8.29 has a very high fitness value (0.99) and not so many deviations. Thus, we proceed to the next step of the approach shown in Figure 8.24: simplifying the model to improve its precision value. In this step, we removed all transitions in the net whose removal does not change the set of all traces that can be produced by the model, i.e., *unnecessary transitions*. For

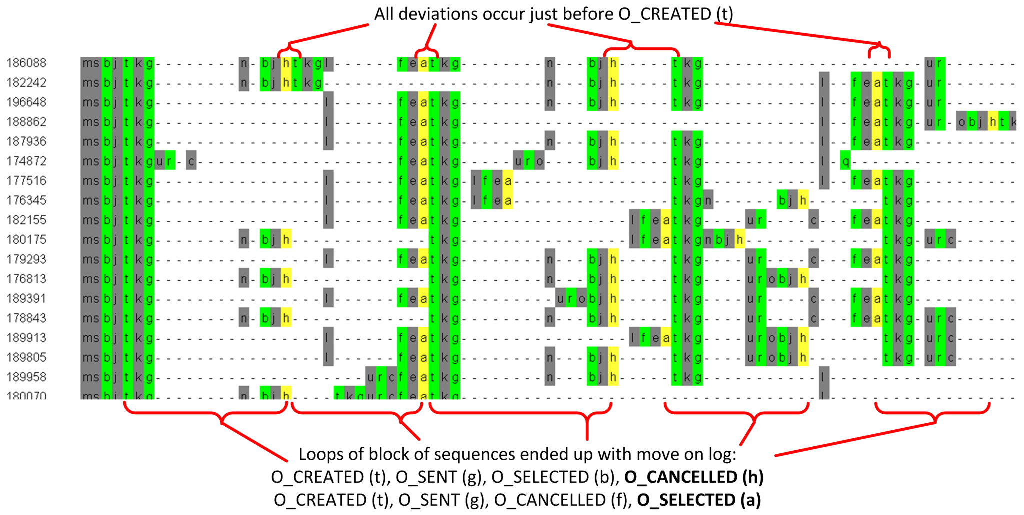


Figure 8.26: Aligned alignments between the traces of the Dutch Financial Institution log and the repaired model in Figure 8.25.

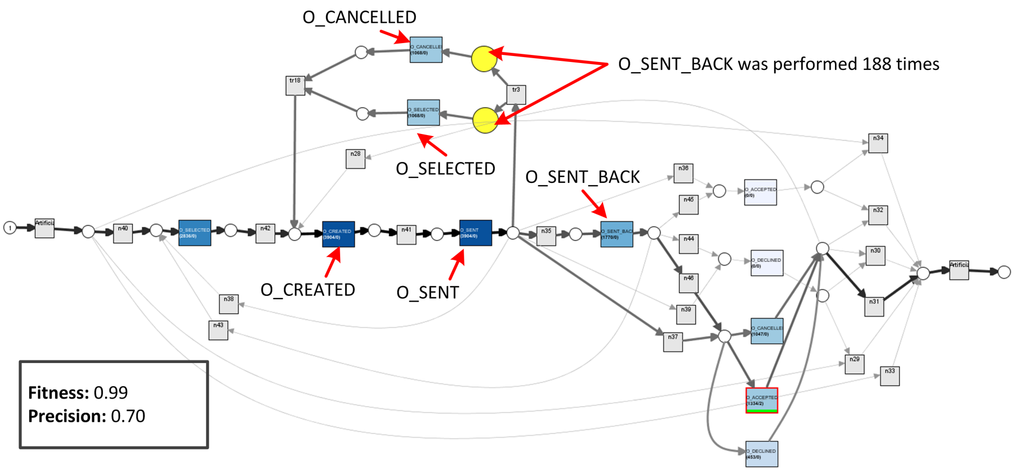


Figure 8.27: The model shown in Figure 8.22, after the second repair iteration.

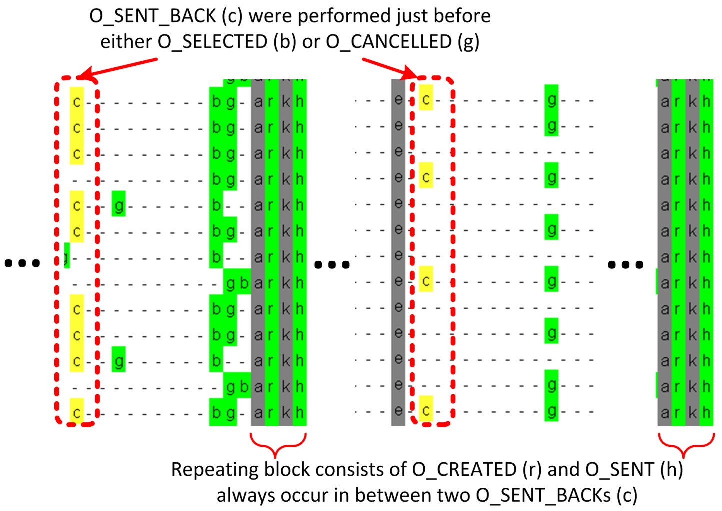

O_SENT_BACK (c) were performed just before either O_SELECTED (b) or O_CANCELLED (g)
Repeating block consists of O_CREATED (r) and O_SENT (h) always occur in between two O_SENT_BACKs (c)

Figure 8.28: Aligned Alignments between the traces of the Dutch Financial Institution log and the repaired model in Figure 8.27.

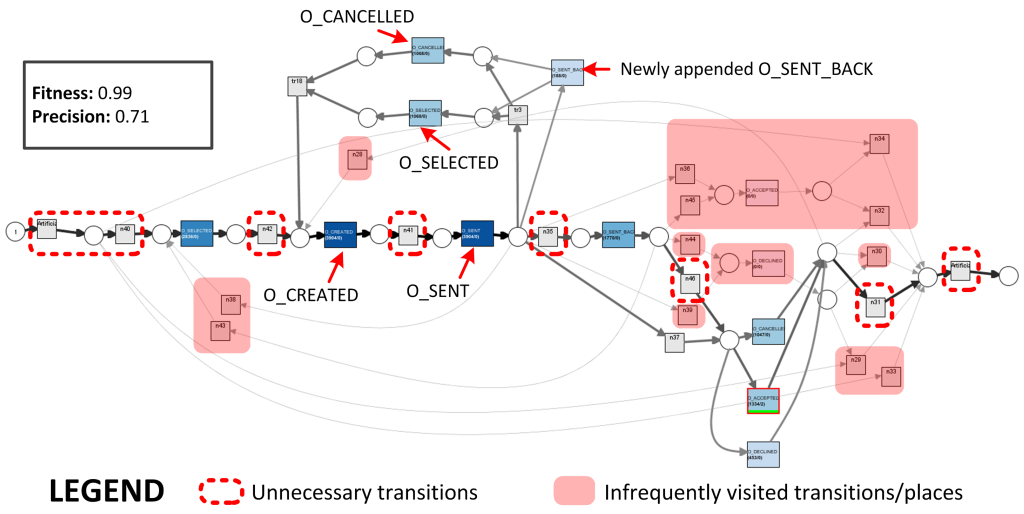


Figure 8.29: The model shown in Figure 8.22, after the third repair.

example, by applying a standard reduction rule on all invisible transitions that have exactly one input and one output arcs we can remove superfluous transitions. Furthermore, we removed transitions/places that were very infrequently visited. Removal of invisible transitions are performed according to Murata's Petri net reduction rules [119].

With such a simplification, we obtain the model shown in Figure 8.30 from the model shown in Figure 8.29. The relative fitness and precision value between the original log and the simplified net in Figure 8.30 are close to perfect (i.e., 1.0). Compare these values to the values of the Petri net originally discovered using heuristic miner in Figure 8.22. This example shows that alignments can also be used to discover better process models, given an event log and an original process model.

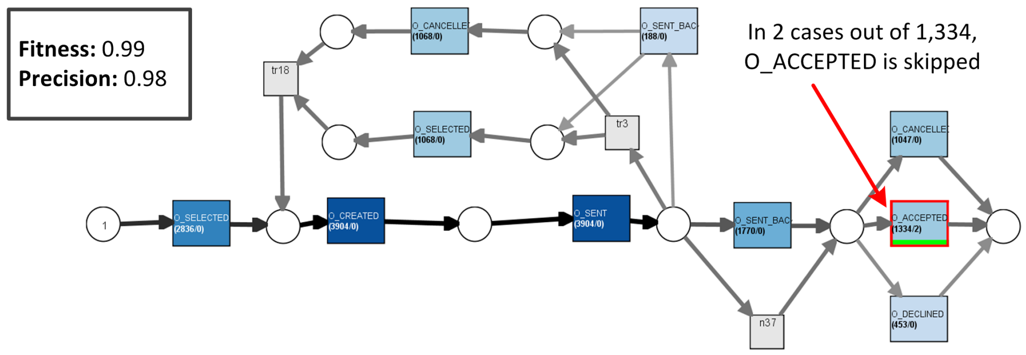


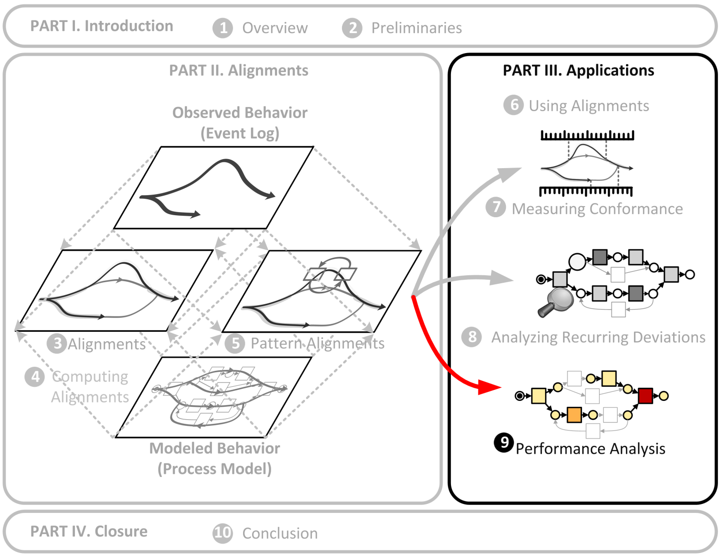
Figure 8.30: The model shown in Figure 8.22, after the final (fourth) repair.

8.5 Conclusion

In a system where process models do not strictly enforce process executions, deviation analysis is important to improve the performance of the overall process. Given an event log, a process model, and an multi-set of alignments for all traces in the log and the model, in this chapter we show how the multi-set can be exploited to analyze possible causes of deviations. We showed various visualizations of the multi-set that yield different insights into deviations. These visualizations are complementary to each other. Furthermore, we showed that the multi-set allows for the application of data-oriented techniques such as data mining to process-oriented analysis, thus broadening the scope of deviation analysis that can be performed to understand deviations and the context where they often occur. Finally, we also showed that the knowledge obtained by performing deviation analysis is not only limited to understanding causes of deviations, but also useful to repair process models to better reflect reality.

Chapter 9

Robust Performance Analysis



9.1 Introduction

Performance measurements of business processes are essential to many approaches that improve performance of organizations such as Total Quality Management [132], Business Process Re-engineering [73], and Six Sigma [123]. Typically, such information is provided by Business Activity Monitoring (BAM) applications that support operational processes. These applications record occurrences of activities and map them to the elements of predefined process models, so that performance can be calculated and projected back onto the models. However, installations of such applications are often costly and non-trivial. Moreover, such applications typically require executions to perfectly fit predefined models, while some degree of flexibility to deviate from them on an operational level is often desired [125, 141, 156].

The abundance of data in information systems allows performance measurements

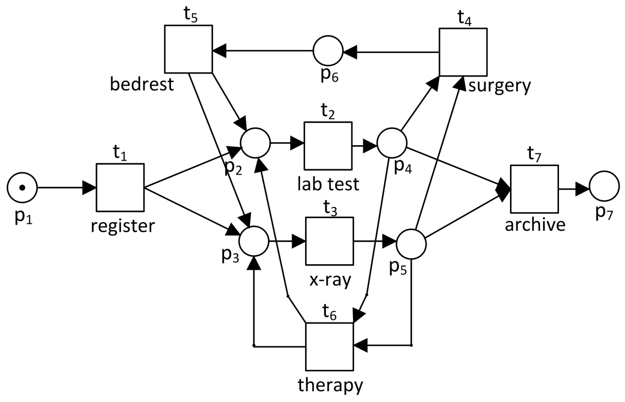


Figure 9.1: A Petri net showing hernia ingualis patients handling process.

based on recorded behavior in the form of event logs. Given an event log and a process model, performance can be measured by replaying the events in the log on the model, e.g., [85]. However, such an approach requires the log to (1) perfectly fit the prescribed model, and (2) be at the same level of granularity as the model. In real life situations, these requirements may not always be satisfied.

Take for example the inguinal hernia patient treatment process in a hospital, shown in Figure 9.1. A patient needs to register him/herself (register) before taking both lab tests and x-ray tests (lab test and x-ray). Based on the test results, a doctor may either suggest the patient to undergo a surgery (surgery) or home therapy (therapy). A patient with therapy treatment must return for another round of examinations a few days after the treatment ends. A patient who has a surgery needs to take a period of rest at the hospital (bedrest) before later going through to another round of examinations. If the examinations show that the patient is healthy, all executed activities are archived (archive) by a hospital administration officer for further reference.

Some of the commonly measured performance metrics are the *service time* of activities (the time spent from the moment a resource starts working on an instance of an activity until the moment the instance is completed), the *waiting time* of activities (the time spent from the moment all non-human resources required to perform an instance of an activity are available until the moment a resource starts working on the activity), and the *sojourn time* of activities (the sum of waiting time and service time of activities). To measure these metrics, events must be logged every time an activity is started and completed [41, 207]. Thus, in this chapter we use the notion of complex event logs and assume that events are recorded each time activities are started/completed.

Given a complex event log and a Petri net, the reconstruction of *activity instances* from events in the log is crucial before performance can be measured. Since the execution of one case typically does not directly influence other cases, activity instances can be reconstructed per case. Figure 9.2 shows a perfectly fitting complex trace for the net shown in Figure 9.1 where events are ordered based on their timestamps. Note that the execution of a transition in the net is recorded as two events, i.e., an instance of an activity is recorded as two events. Thus, the net is at a higher level of granularity than the logged complex events.

To construct activity instances from the trace and the net, first we convert the net at a high level of granularity into a net at a low level of granularity that allows for exactly

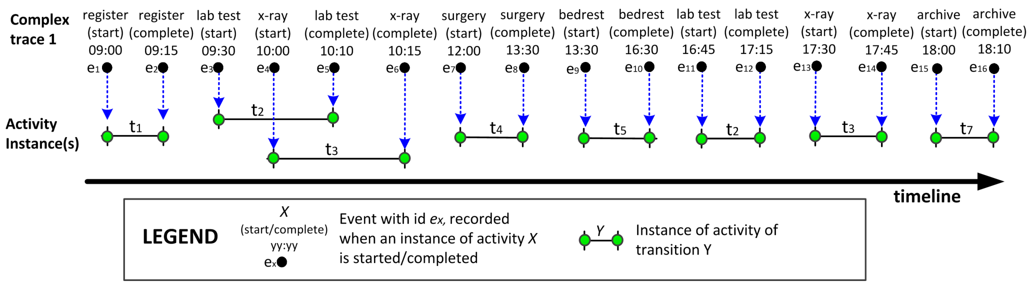

Complex trace 1
register (start) 09:00 e1
register (complete) 09:15 e2
lab test (start) 09:30 e3
x-ray (start) 10:00 e4
lab test (complete) 10:10 e5
x-ray (complete) 10:15 e6
surgery (start) 12:00 e7
surgery (complete) 13:30 e8
bedrest (start) 13:30 e9
bedrest (complete) 16:30 e10
lab test (start) 16:45 e11
lab test (complete) 17:15 e12
x-ray (start) 17:30 e13
x-ray (complete) 17:45 e14
archive (start) 18:00 e15
archive (complete) 18:10 e16
Activity Instance(s)
t1
t2
t3
t4
t5
t2
t3
t7
timeline
LEGEND
X (start/complete) yy:yy ex
Event with id ex, recorded when an instance of activity X is started/completed
Y
Instance of activity of transition Y

Figure 9.2: Example of a set of activity instances constructed from a perfectly fitting complex trace with respect to the net in Figure 9.1.

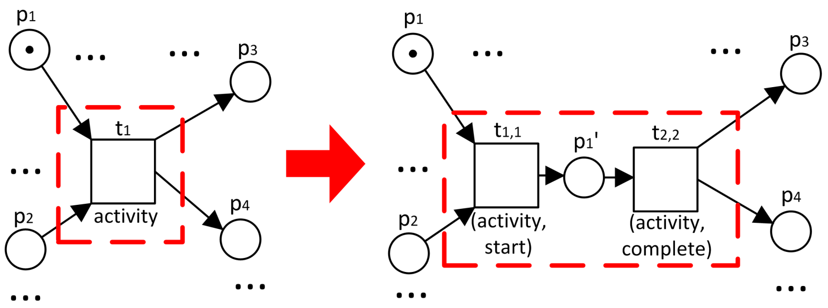


Figure 9.3: An illustration of how a transition in a Petri net at a high level of granularity (left) is decomposed into a transition at a lower level of granularity (right). A low level transition is labeled by the corresponding name and the life-cycle transaction type (i.e., start/complete).

the same behavior. We use a similar approach as the pattern-based deviation analysis in Chapter 5 to perform such a conversion. A full life-cycle of an instance of an activity is recorded in a complex event log as a pair of events $e_1, e_2 \in \mathcal{E}$ that refer to the same activity with different life-cycle transactions (“start” and “complete”). The event in the pair that should occur first has a life-cycle transaction type “start” while the other event that should occur later has a life-cycle transaction type “complete”. A transition of a Petri net at a high level of granularity is therefore decomposed into a sequence of two transitions at a low level of granularity as shown in Figure 9.3. The transition that starts the sequence is labeled with a pair of activity names and “start” life-cycle transaction type, while the other transition is labeled with the pair of the same activity name and “complete” life-cycle transaction type. For convenience, in the remainder of this chapter we use *high level nets* and *low level nets* to refer to Petri nets at a high level of granularity and Petri nets at a low level of granularity respectively.

Furthermore, we need to distinguish the events that mark the start of activity instances from the events that mark the completion of activity instances. Thus, in the rest of this chapter we use a classifier that maps complex events to a combination of the activity name and the life-cycle transaction, such that for all events $e \in \mathcal{E} : \underline{e} = (\#_{act}(e), \#_{trans}(e))$. Recall that classifiers determine the way events are named (see Definition 2.3.3). For example, the names of the first and second events in the complex trace of Figure 9.2 are $(register, start)$ and $(register, complete)$ respectively. This way, events in Figure 9.2 can be trivially mapped to low level transitions with the same label in Figure 9.4.

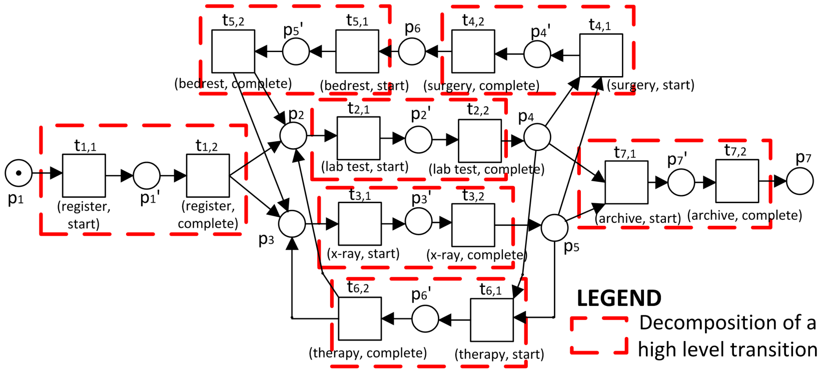


Figure 9.4: A low level Petri net of the net shown in Figure 9.1, assuming that an instance of an activity consists of a pair of events each with a life-cycle transaction "start" and "complete" respectively.

The complex trace in Figure 9.2 perfectly fits the low level net. Thus, activity instances can be constructed by pairing each event in the trace mapped to the "start" transition of an activity with its first succeeding event which is mapped to the "complete" transition of the same activity. In the example shown in Figure 9.2, there is only one instance for each of the activities register, lab test, bedrest, and archive. There are two instances for each of the activities lab test and x-ray. From the identified activity instances, performance metrics can trivially be measured. For instance, by taking into account both instances of activity lab test we can measure its average service time: $\frac{1}{2} \cdot (40 + 30) = 35$ minutes.

Consider the non fitting complex trace in Figure 9.5, obtained by removing four events from the original trace shown in Figure 9.2. In this example, naively pairing events in the same way as mentioned before yields a misleading instance for activity lab test (see Figure 9.5(a)). As shown by the set of constructed instances in the figure, an instance of lab test was not completed yet when instances of both surgery and bedrest were started, while this is not allowed according to the net. In this case, the average service time of activity lab test from the identified instance is 7.25 hours. This value is much higher than the value previously obtained from activity instances of the perfectly fitting trace (35 minutes). Pairing the events backwards, i.e., each event whose life-cycle transaction type is "complete" is paired with its preceding event whose life-cycle transaction type is "start" and refers to the same activity, yields another problem as shown in Figure 9.5(a). According to the figure, the service time of activity x-ray is 7 hours 45 minutes, which is much longer than the average service time of the same activity measured from the activity instances in Figure 9.2.

To identify activity instances from the log accurately, we construct a multi-set of alignments from the log and the low level net. Activity instances are then constructed from the multi-set. Note that given multiple alignments of the same complex trace in the log, the performance values measured from one alignment may be different than the performance values measured from other alignments. To avoid this problem, *we use a basic oracle function when constructing such multi-set of alignments*, i.e., for each trace in an event log, there is only one alignment in the multi-set.

Section 9.2 presents related work on process performance measurement. Section 9.3 describes an approach to discover a set of activity instances from a given complex event log and a Petri net and how to use them in order to measure performance. The proposed

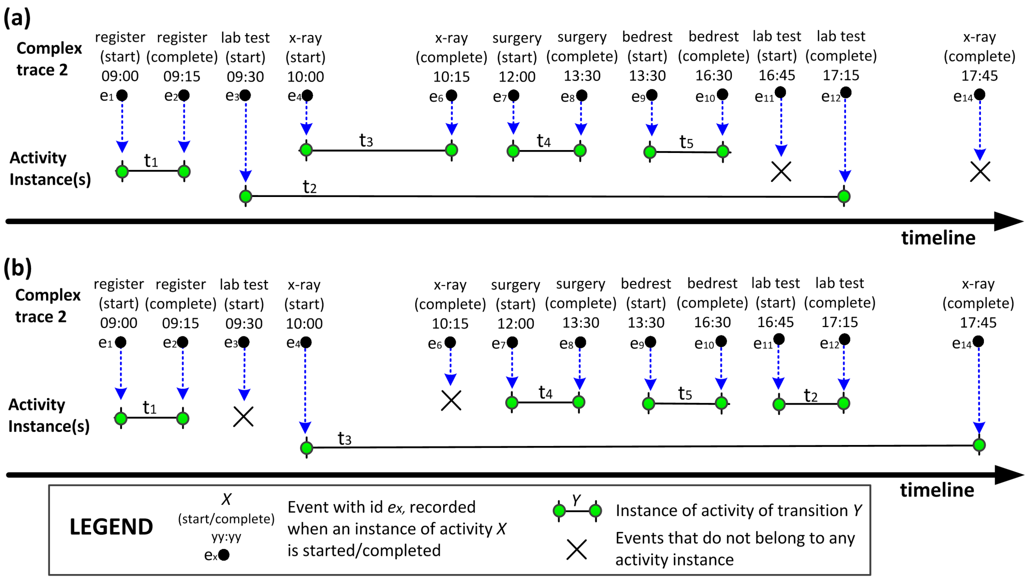

(a)
Complex trace 2
register (start) 09:00 e1
register (complete) 09:15 e2
lab test (start) 09:30 e3
x-ray (start) 10:00 e4
x-ray (complete) 10:15 e6
surgery (start) 12:00 e7
surgery (complete) 13:30 e8
bedrest (start) 13:30 e9
bedrest (complete) 16:30 e10
lab test (start) 16:45 e11
lab test (complete) 17:15 e12
x-ray (complete) 17:45 e14
Activity Instance(s)
t1
t2
t3
t4
t5
timeline
(b)
Complex trace 2
register (start) 09:00 e1
register (complete) 09:15 e2
lab test (start) 09:30 e3
x-ray (start) 10:00 e4
x-ray (complete) 10:15 e6
surgery (start) 12:00 e7
surgery (complete) 13:30 e8
bedrest (start) 13:30 e9
bedrest (complete) 16:30 e10
lab test (start) 16:45 e11
lab test (complete) 17:15 e12
x-ray (complete) 17:45 e14
Activity Instance(s)
t1
t4
t5
t2
t3
timeline
LEGEND
X (start/complete) yy:yy ex
Event with id ex, recorded when an instance of activity X is started/completed
Y
Instance of activity of transition Y
Events that do not belong to any activity instance

Figure 9.5: Examples of misleading activity instances constructed from a non-fitting complex trace for the net shown in Figure 9.1 by pairing events with the same activity attribute (a) from the start of the trace forward, and (b) from the end of the trace backward.

approach is evaluated in Section 9.4, and Section 9.5 concludes this chapter.

9.2 Related Work

Process performance measurements are crucial to support organizations in making strategic decisions [106, 131, 207]. Various process performance metrics have been defined in literature (e.g., [52, 144, 186, 213]). Despite numerous case studies showing that deviations to predefined models often occur in reality [8, 65, 151, 180], not much attention have been given to measure performance in systems where deviations occur. Zur Muehlen and Rosemann [212] proposed a prototype process analysis tool PISA to analyze both technical and organizational performance based on event logs from multiple workflow management systems. However, the approach did not take into account deviating events. Similarly, Castellanos et al. [31] proposed the iBOM framework to measure performance of processes in abstract views, but it did not specify any approach to deal with possible deviations. Similar drawbacks are also found with other performance analysis approaches that rely on activity monitoring tools, such as [41, 206, 207].

Popova and Sharpanskykh [131] proposed a formal approach to evaluate organizational performance and take into account that people may deviate from prescribed process models. However, in cases where such deviations occur, manual inspection is required to determine the reason behind the deviations. This limits the applicability of the approach. Given an event log and a low level net, Hornix [85] proposed an approach to measure performance metrics by replaying each trace in the log on the net using the token-based replay approach [151]. However, as shown in Chapter 3, the token-based replay approach yields misleading results when the net has duplicate/invisible transitions. Nakatumba and Van der Aalst [120] proposed an approach to identify outliers and estimating activity start events, given an event log. The approach, however, does not

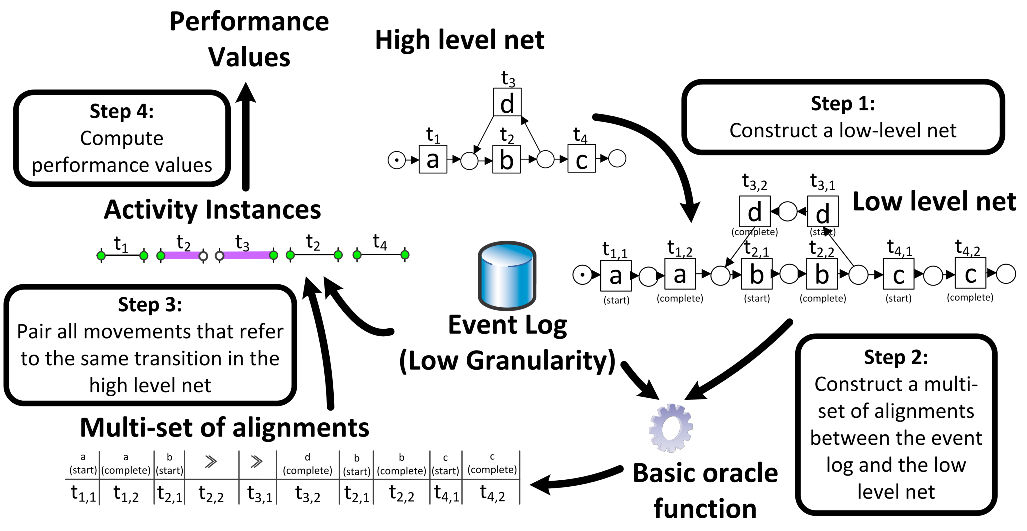

Performance Values
High level net
Step 1: Construct a low-level net
Step 4: Compute performance values
Low level net
Activity Instances
Step 3: Pair all movements that refer to the same transition in the high level net
Event Log (Low Granularity)
Step 2: Construct a multi-set of alignments between the event log and the low level net
Multi-set of alignments
Basic oracle function

Figure 9.6: Overview of an alignment-based approach to measure performance by first constructing activity instances.

take into account any process model.

Computing a mapping between events at a low level of granularity and a high level net is an implicit problem that also needs to be addressed in performance measurement from event logs. In the absence of an explicit mapping between a set of events and a high level net, a low level net can be discovered from the events together with a mapping between the elements of the net and the high level net. Given an event log at a low level of granularity and a high level net in form of a hierarchical markov model, Ferreira et al. [63] proposed a method based on Expectation-Minimization techniques to discover a low level net and a mapping between the net to the high level net. Nevertheless, this approach does not guarantee that all events can be mapped to elements of the low level net, while this could be crucial in performance measurements. In [120], Nakatumba et al. proposed an approach to remove outliers and estimate the start events of activity instances in noisy event logs. This approach can be used as a pre-processing step before measuring performance using event logs.

9.3 Measuring Performance

Related work mentioned in Section 9.2 shows that there is no approach that is able to robustly measure performance in systems where deviations occur. Given a complex event log and a high level net, we propose an approach to construct activity instances based on alignments. As mentioned in Section 9.1, in this chapter we assume that events are recorded when activity instances are started and completed. Thus, for all events in the event log, there are two possible values of transaction type attribute: “start” or “complete”, i.e., let $\mathcal{E}$ be the complex event universe, for all events $e \in \mathcal{E} : \#_{trans}(e) \in \{start, complete\}$. All events in the log with transaction type attribute other than “start” and “complete” are filtered out.

Figure 9.6 shows an overview of all steps proposed in this section to measure performance based on an event log and a given high level net where events were recorded

Table 9.1: A complex event log of the net shown in Figure 9.1, consists of only one case.

Case id	Event id	Properties		
		Timestamp	**Activity**	**Lifecycle**
c_1	e_1	9:00	register	start
	e_2	9:15	register	complete
	e_3	10:00	x-ray	start
	e_4	10:10	lab test	start
	e_5	10:15	x-ray	complete
	e_6	12:00	therapy	start
	e_7	13:30	therapy	complete
	e_8	18:00	archive	start
	e_9	18:10	archive	complete

e_1	e_2	e_3	e_4	e_5	$\gg$	e_6	e_7	e_8	e_9
(register, start)	*(register, complete)*	*(x-ray, start)*	*(lab test, start)*	*(x-ray, complete)*	$\gg$	*(therapy, start)*	*(therapy, complete)*	*(archive, start)*	*(archive, complete)*
(register, start)	*(register, complete)*	*(x-ray, start)*	*(lab test, start)*	*(x-ray, complete)*	*(lab test, complete)*			*(archive, start)*	*(archive, complete)*
$t_{1,1}$	$t_{1,2}$	$t_{3,1}$	$t_{2,1}$	$t_{3,2}$	$t_{2,2}$	$\gg$	$\gg$	$t_{7,1}$	$t_{7,2}$

Figure 9.7: The top row: Events of case c_1, taken from the complex event log in Table 9.1. $\gg$ denotes movements that do not relate to any events of case c_1, **The second row until bottom**: An optimal alignment between the trace of the complex trace of c_1 in Table 9.1 and the net shown in Figure 9.4 with respect to the standard likelihood cost function.

when activity instances are either started or completed. The first two steps are performed to relate the events of the log to the transitions of the net. First, a Petri net with the same level of granularity as the events in the log is constructed (i.e., a low level net). The constructed low level net must only allow for the same set of traces as yielded by the high level net. Then, we construct a multi-set of alignments of the log with respect to the previously constructed net using a basic oracle function to explicitly show deviations between events in the log and the low level net. The multi-set also shows the behavior allowed by the net that is most likely represents the behavior observed in log, which we then exploit to construct activity instances. Performance metrics are then computed based on the identified activity instances.

Both examples in Figure 9.5 show that identifying deviations is crucial before measuring performance. In Section 9.1, we already showed how to construct the low level net from a high level net. Given a complex event log, a low level net, and a basic oracle function, we construct an alignment for each trace in the log using the oracle function. Take for example the complex event log of the high level net in Figure 9.1, shown in Table 9.1. Suppose that we use a basic and optimal oracle function with respect to the standard likelihood cost function. Figure 9.7 shows an optimal alignment between the trace of case c_1 and the low level net in Figure 9.4 yielded by the oracle. According to the alignment, an event that indicates the completion of an instance of activity lab test is missing in the trace. Furthermore, two events of the same activity therapy are not supposed to be in the trace according to the net.

Given an alignment between a trace and a low level net, we construct instances of activities by pairing all movements that refer to the same high level transition from the start of the alignment to its end. We ignore all moves on log and only take into account both synchronous moves and moves on model because all move on logs do not refer to any high level transition. Note that an instance of an activity is not necessarily derived

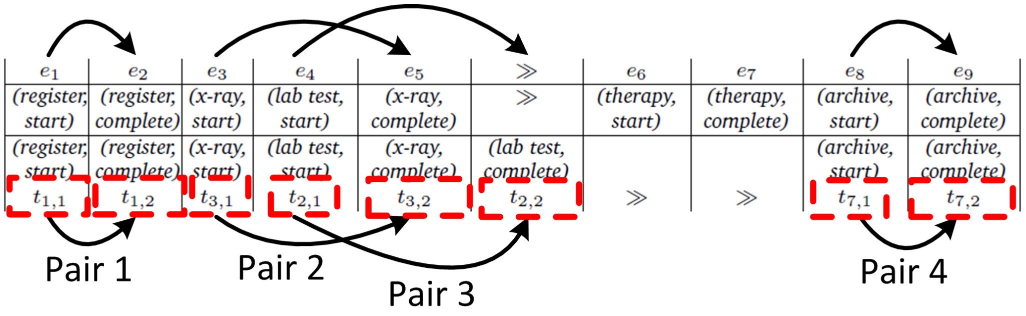

e_1 e_2 e_3 e_4 e_5 $\gg$ e_6 e_7 e_8 e_9
(register, start) (register, complete) (x-ray, start) (lab test, start) (x-ray, complete) $\gg$ (therapy, start) (therapy, complete) (archive, start) (archive, complete)
(register, start) (register, complete) (x-ray, start) (lab test, start) (x-ray, complete) (lab test, complete) (archive, start) (archive, complete)
$t_{1,1}$ $t_{1,2}$ $t_{3,1}$ $t_{2,1}$ $t_{3,2}$ $t_{2,2}$ $\gg$ $\gg$ $t_{7,1}$ $t_{7,2}$
Pair 1
Pair 2
Pair 3
Pair 4

Figure 9.8: Pairing each low level transition whose transaction life-cycle label is "start" with its closest successor transition whose transaction life-cycle label is "complete" and refers to the same activity from the alignment shown in Figure 9.7.

from two synchronous moves. An activity instance may be derived from a move on model and a synchronous move, or two moves on model.

Take for example case c_1 in the log of Table 9.1 and the alignment of Figure 9.7 between the trace of c_1 and the low level net shown in Figure 9.4. We pair each move on model/synchronous move in the alignment whose transition is labeled with transaction life-cycle "start" with its closest successor that: (1) refers to the same high level transition, and (2) has a transaction life-cycle label "complete". Figure 9.8 shows how the movements of the alignment shown in Figure 9.7 are paired. Each pair of movements yields an activity instance.

We denote an instance of an activity a as a tuple of three elements (x, y, z), where both x and y are either events corresponding to a or $\gg$, and z is a high level transition labeled a. If x is an event then its transaction life-cycle is "start". Similarly, if y is an event then its transaction life-cycle is "complete". For example, the four activity instances discovered in Figure 9.8 are (e_1, e_2, t_1), (e_3, e_5, t_3), $(e_4, \gg, t_2)$, and (e_8, e_9, t_7). We say that $(e_4, \gg, t_2)$ is a *partial complete instance* because it does not contain any event that marks its completion, i.e., it is replaced with $\gg$. Similarly, an activity instance $(x, \gg, z)$ is a *partial start instance* if x is an event. An activity instance $(\gg, \gg, z)$ without both start and complete events is called a *missing instance*, while an activity instance with both start and complete events is a *perfect instance*.

Figure 9.9 shows another visualization of the discovered activity instances in Figure 9.8. Note that both events e_6 and e_7 are not considered to be a part of any activity instance as they are involved in moves on log according to the alignment shown in Figure 9.7. Applying the same approach to the complex trace shown in Figure 9.12 using the same oracle function yields a set of activity instances shown in Figure 9.10. Note that the set of discovered instances in Figure 9.10 resembles the one shown in Figure 9.2.

Given a complex event log, a low level Petri net, and a basic oracle function, we repeat the same steps for all cases in the log to obtain a set of activity instances for each case in the log. This set is used further to measure performance.

The last step of the approach proposed in Figure 9.6 is to compute performance values based on the discovered activity instances. Given a set of activity instances, performance can be computed in a trivial way. The service time of an activity can be computed from all perfect instances of the activity. Take for example the discovered activity instances in

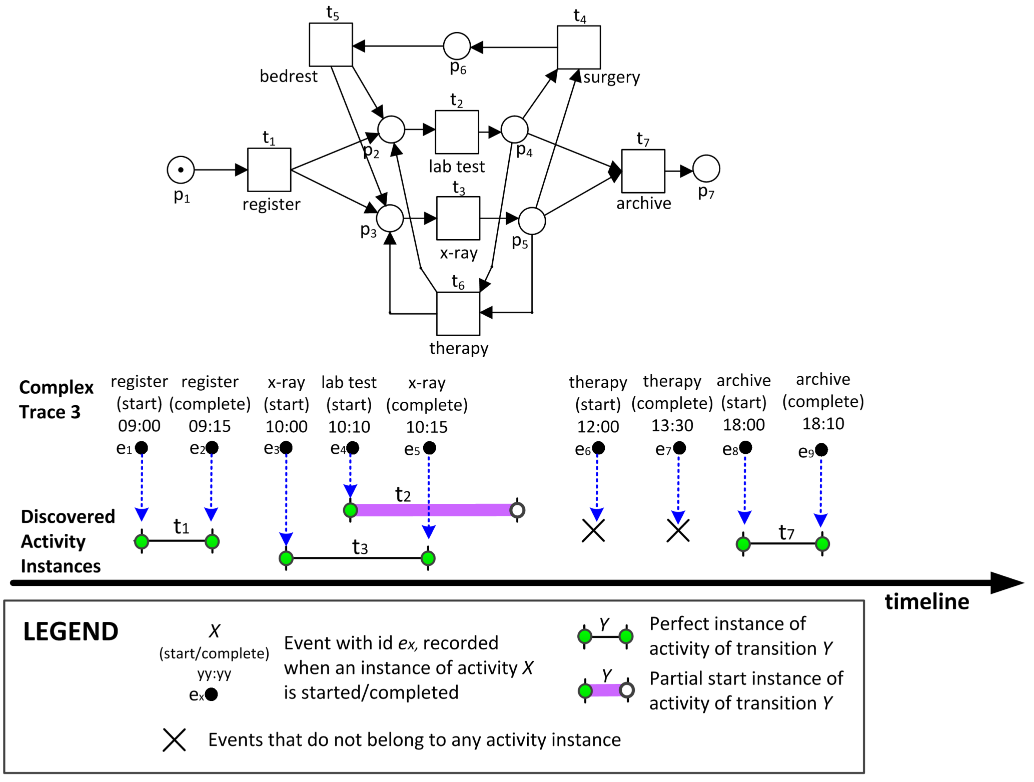

t5
bedrest
p6
t4
surgery
t2
lab test
p2
p4
t1
register
p1
t3
x-ray
p3
p5
t7
archive
p7
t6
therapy
Complex Trace 3
register (start) 09:00 e1
register (complete) 09:15 e2
x-ray (start) 10:00 e3
lab test (start) 10:10 e4
x-ray (complete) 10:15 e5
therapy (start) 12:00 e6
therapy (complete) 13:30 e7
archive (start) 18:00 e8
archive (complete) 18:10 e9
Discovered Activity Instances
t1
t2
t3
t7
timeline
LEGEND
X (start/complete) yy:yy ex
Event with id ex, recorded when an instance of activity X is started/completed
Y Perfect instance of activity of transition Y
Y Partial start instance of activity of transition Y
Events that do not belong to any activity instance

Figure 9.9: Top: The high level net shown in Figure 9.1, **Bottom:** The discovered activity instances by pairing the movements of alignments shown in Figure 9.7.

Figure 9.11. The service time of activity x-ray can be computed from the time required to perform the perfect activity instance (e_3, e_5, t_3), which is 15 minutes. Assuming that all non human resources to perform an activity are available immediately whenever all transitions labeled with the activity are also enabled in the net, the average waiting time of an activity can be measured from a pair of activity instance X and Y where: (1) X is either a perfect or partial start instance of the activity, and (2) Y is either perfect or partial complete instance of other activities that directly precede X. In this example, the waiting time of activity x-ray is 45 minutes (i.e., the time spent between the completion of (e_1, e_2, t_1) and the start of (e_3, e_5, t_3). The sojourn time of the activity x-ray in this example can be measured from the moment (e_1, e_2, t_1) is completed until the moment (e_3, e_5, t_3) is completed, which is 1 hour.

Performance can also be measured from partial start/complete activity instances. For example, no perfect instance of lab test can be discovered in Figure 9.11. However, the waiting time of activity lab test can still be measured from the moment of (e_1, e_2, t_1) was completed until the moment $(e_4, \gg, t_2)$ was started (55 minutes).

Identifying activity instances is a crucial step in measuring performance. In cases where some performance metrics cannot be measured because of deviations, one can exploit the diagnostics provided by the discovered instances in order to estimate performance values based on statistical analysis. For example, the service time of activity lab test cannot be measured from the discovered activity instances in Figure 9.11 because the moment the instance $(e_4, \gg, t_2)$ was completed is unknown. However, we know that activity archive can only occur after an instance of lab test is completed according to the net

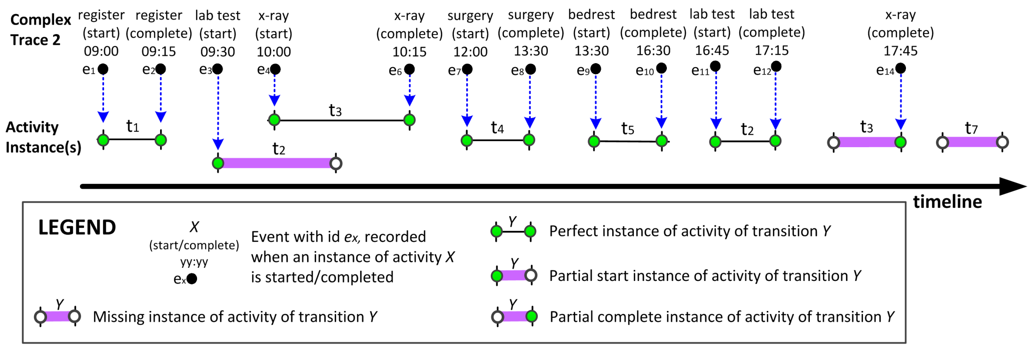

Complex Trace 2
register (start) 09:00 e1
register (complete) 09:15 e2
lab test (start) 09:30 e3
x-ray (start) 10:00 e4
x-ray (complete) 10:15 e6
surgery (start) 12:00 e7
surgery (complete) 13:30 e8
bedrest (start) 13:30 e9
bedrest (complete) 16:30 e10
lab test (start) 16:45 e11
lab test (complete) 17:15 e12
x-ray (complete) 17:45 e14
Activity Instance(s)
t1
t2
t3
t4
t5
t2
t3
t7
timeline
LEGEND
X (start/complete) yy:yy ex
Event with id ex, recorded when an instance of activity X is started/completed
Perfect instance of activity of transition Y
Partial start instance of activity of transition Y
Missing instance of activity of transition Y
Partial complete instance of activity of transition Y

Figure 9.10: The set of activity instances discovered by applying the approach in Figure 9.6 to the non-fitting complex trace in Figure 9.5 and the high-level net in Figure 9.1.

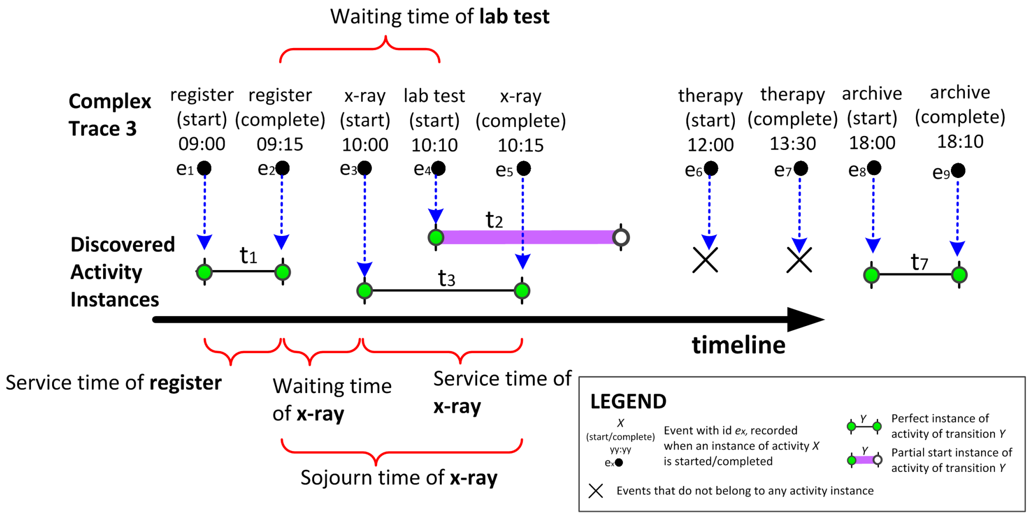

Waiting time of lab test
Complex Trace 3
register (start) 09:00 e1
register (complete) 09:15 e2
x-ray (start) 10:00 e3
lab test (start) 10:10 e4
x-ray (complete) 10:15 e5
therapy (start) 12:00 e6
therapy (complete) 13:30 e7
archive (start) 18:00 e8
archive (complete) 18:10 e9
Discovered Activity Instances
t1
t2
t3
t7
timeline
Service time of register
Waiting time of x-ray
Service time of x-ray
Sojourn time of x-ray
LEGEND
X (start/complete) yy:yy ex
Event with id ex, recorded when an instance of activity X is started/completed
Perfect instance of activity of transition Y
Partial start instance of activity of transition Y
Events that do not belong to any activity instance

Figure 9.11: Some performance metrics that can be measured from the discovered activity instances shown in Figure 9.9.

in Figure 9.1. Thus, we know that the completion time of instance $(e_4, \gg, t_2)$ must fall between the start time of the instance (10:10) and the start time of instance (e_8, e_9, t_7) (18:00). One may take a naive approach to assume that all partial start/complete instances were performed instantly [7]. This way, instance $(e_4, \gg, t_2)$ is assumed to be completed at time 10:10. Similarly, one can also assume that the instances shown in Figure 9.11 is correct, i.e., $(e_4, \gg, t_2)$ ended between 10.15 (e_5) and 12.00 (e_6). One may also perform statistical analysis to better estimate the time of completion of the instance if more information is available, e.g., resources [120]. In this case, we choose to make no assumptions on unknown event timestamps, i.e., all unknown events are ignored. In Section 9.4, we report the experimental results to identify the most appropriate approach.

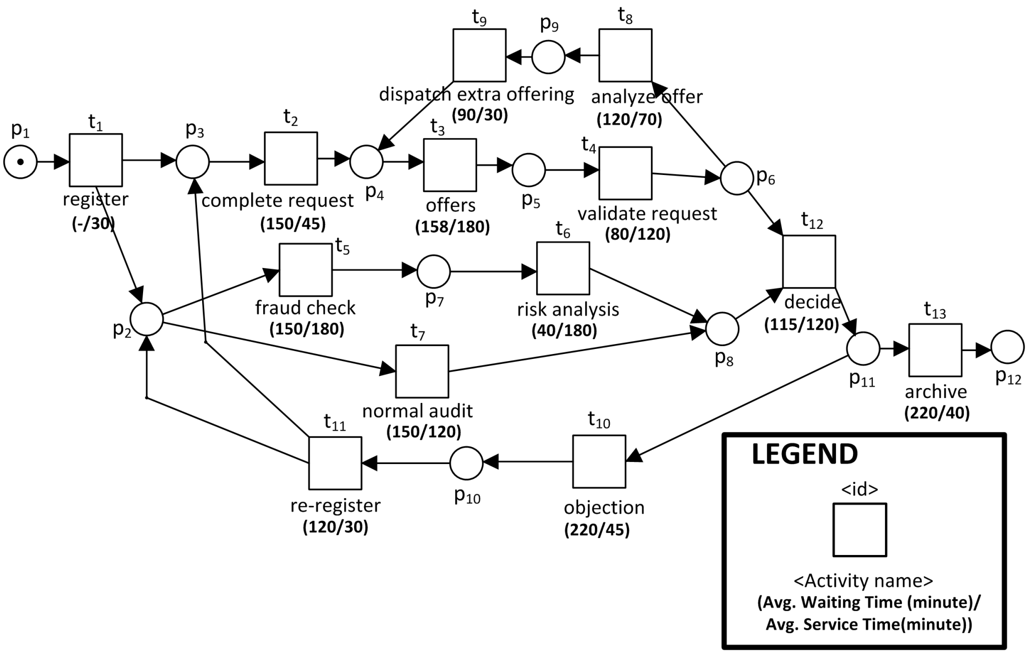


Figure 9.12: A personal loan application process in a Petri net terms.

9.4 Experiments

To evaluate the proposed approach in this chapter, we implemented it using the ProM framework. A set of experiments was conducted using a Petri net and a set of artificial logs with real-life complexity. Furthermore, a case study using an event log and a Petri net taken from a municipality in the Netherlands was performed to evaluate the applicability of the approach in real-life settings. The results of the experiments with the artificial model and logs are explained in Section 9.4.1 and the results of the case study are reported in Section 9.4.2.

9.4.1 Artifical Logs and Models

We created a Petri net and a set of artificial event logs of real-life complexity loosely based on the application for an online personal loan process of a Dutch Financial Institute [192]. The net is shown in Figure 9.12. It consists of 13 transitions where each transition is labeled with a unique activity.

An applicant needs to first register for a loan application request (register). Then, an officer categorizes the request, marks it as a completed request, and notifies the applicant about the acceptance of the application for further processing (complete request). Then, a public relation officer sends some information about the types of loans that match the application request and a validation request form (offers). Another officer then evaluates the request according to a set of predefined rules (validate request). In parallel, an internal auditor decides whether the request only needs to be checked either normally (normal audit) or thoroughly to prevent possible fraud (fraud check). An application that undergoes fraud checking is analyzed by a risk analyst (risk analysis). Both the risk analysis and the normal audit report are sent to an application manager. The manager then decides to either grant the application request or to reject it (decide). Soon after a decision is made, it is sent to the applicant. If the applicant is a potential client for some new loan products, the manager may also decide postpone his decision and assign another analyst to create a tailored offering for the applicant (analyze offer). The tailored offering is sent back to the public relation officer (dispatch extra offering) to be offered to the applicant. If there is no objection to a decision made by the manager for the applicant, the whole application request history is archived (archive). However, if the applicant objects (objection), then the application is re-registered for another round of evaluations together with additional notes of objections (re-register).

We generated 30 complex event logs using the CPN Tools [139], each log consists of 1,000 cases. Two events were generated each time a transition labeled with an activity is fired: one event marks the start of an instance of the activity and the other event marks the completion of the instance. The number of events per case varies from 14 events per trace to more than 150 events per trace. The service time and the waiting time of all activities were configured to follow exponential distributions. The average service time and the average waiting time per activity are shown in Figure 9.12. In addition to the service time and the waiting time of all activities, we also measured the time spent to execute pairs of activities in sequences: the time spent between the pair of activities analyze offers-dispatch extra offering, fraud check-risk analysis, objection-re-register, and offers-validate request.

Three sets of non-fitting logs were generated from the perfectly fitting logs, each set consists of 30 logs of 1,000 cases. The first set of logs consists of non-fitting logs generated by *swapping activity names* of events randomly. The second set of logs contains non-fitting logs due to random *removal* of events. The third set of logs contains non-fitting logs due to random *addition* of new events to the original logs. We measured the service time and the waiting time of all activities using the non-fitting logs and compared the results to baseline values obtained from measuring the same metrics on perfectly fitting logs. We compared the proposed approach in this chapter against the token-based replay approach that measures the same performance metrics from a given event log and a Petri net [85], implemented in the ProM 5 framework.[1] The token-based replay approach measures performance only from events that can be replayed without missing tokens.

We evaluated possible variations in the approach proposed in this chapter using three different basic and optimal oracle functions. Each function uses a different likelihood cost function. In the remainder of this chapter, for all $x, y \in I\!R : x \ll y$ denotes that the value of x is much less than the value of y and $x \gg y$ denotes that the value of x is much higher than the value of y. Note that we overload notation as before "$\gg$" was used to represent a "no move". The three likelihood cost functions are (1) a cost function where the likelihood cost of moves on model is much less than the likelihood cost of moves on log (MM $\ll$ ML), (2) a cost function where the likelihood cost of moves on model is much higher than the likelihood cost of moves on log (MM $\gg$ ML), and (3) a cost function where the likelihood cost of moves on log is the same as the likelihood cost of moves on model (MM = ML). We selected three different arbitrary likelihood cost functions: MM:ML = 1:50, MM:ML = 50:1, and MM:ML = 1:1 to be referred to as MM $\ll$ ML, MM $\gg$ ML, and MM = ML respectively.

Finally, we compared the accuracy of the proposed approach with an alternative approach where the timestamps of all missing events are injected (i.e., timestamps are

[1]see `http://www.processmining.org`

injected to all moves on model). We used the timestamps injection approach proposed in [7] as a baseline. The timestamp of a missing event in an activity instance is the timestamp of its closest preceding event whose activity instance directly precedes the instance. This way, the service time of a non perfect activity instance is 0. If there is no such event, the timestamp is taken from the closest successor event whose activity instance succeeds the instance. We measured the Mean Square Error (MSE) of all measurements to compare all approaches in the experiments.

Figure 9.13 shows the results of the experiments, presented in the form of scatter plots with polynomial trendlines. The x-axis of the scatter plots shows the weighted fitness values and the y-axis shows the MSE of performance measurements. As shown by the scatter plots, all approaches provide accurate performance measurements when measuring performance based on perfectly fitting logs (i.e., logs whose weighted fitness value are 1.00). Inaccurate measurements are obtained only in the experiments with logs whose weighted fitness values are below 1.00. This implies that poor fitness values can be an indication of poor performance estimation. Therefore, one needs to take into account weighted fitness values when measuring performance from a given log and a Petri net. We also see that for each approach, given an event log and a process model, there is a limit to the accuracy of the performance estimation it can provide when the log is not perfectly fitting. Beyond this limit, the approach may provide high MSE values.

As shown in Figure 9.13, the token-based approach provides significantly higher MSE measurements for non fitting logs than the alignment-based approaches. This is because the approach does not consider the most-likely-followed paths according to the original traces in the logs when measuring performance. In contrast, the alignment-based approaches try to guess a "correct" path followed by each trace in the logs before measuring performance. In many cases, this leads to more accurate performance measurements.

As expected, performance measurements gets more accurate when more events in the logs are taken into account (see Figure 9.13). The maximum MSE value of all performance metric measurements using the logs with the same fitness level increases from the left to right figures (i.e., from the experiments with likelihood cost function MM $\ll$ ML to the experiments with likelihood cost function MM $\gg$ ML). In the presence of a non-fitting event in a trace, an approach based on cost function MM $\gg$ ML "prefers" discarding the event (i.e., do a move on log) to performing moves on model until the event is enabled. Thus, many moves on log are constructed and therefore many events are discarded during performance measurements. In contrast, the alignment-based approaches that use likelihood cost function MM $\ll$ ML (see the left side of Figure 9.13) prefer alignments with less number of moves on log because the cost of a move on log is much higher than the cost of a move on model. Therefore, more events in the logs are taken into account when measuring performance. This leads to better performance measurement accuracy.

Experiments also show that injecting timestamps naively yields poor performance estimations. The scatter plots on the left of Figure 9.13 show that the MSEs of performance measurements taken without timestamps injections are lower than the MSEs of the same measurements taken with timestamps injections. Many of the alignments constructed with likelihood cost function MM $\ll$ ML have subsequences of moves on model to enable non-fitting events. Naively injecting timestamps to these moves on model yields misleading performance measurements because the same timestamp is assigned to all of them. In contrast, the effect of timestamps injection is negligible in the experiments with likelihood cost function MM $\gg$ ML. As mentioned before, in the experiments with

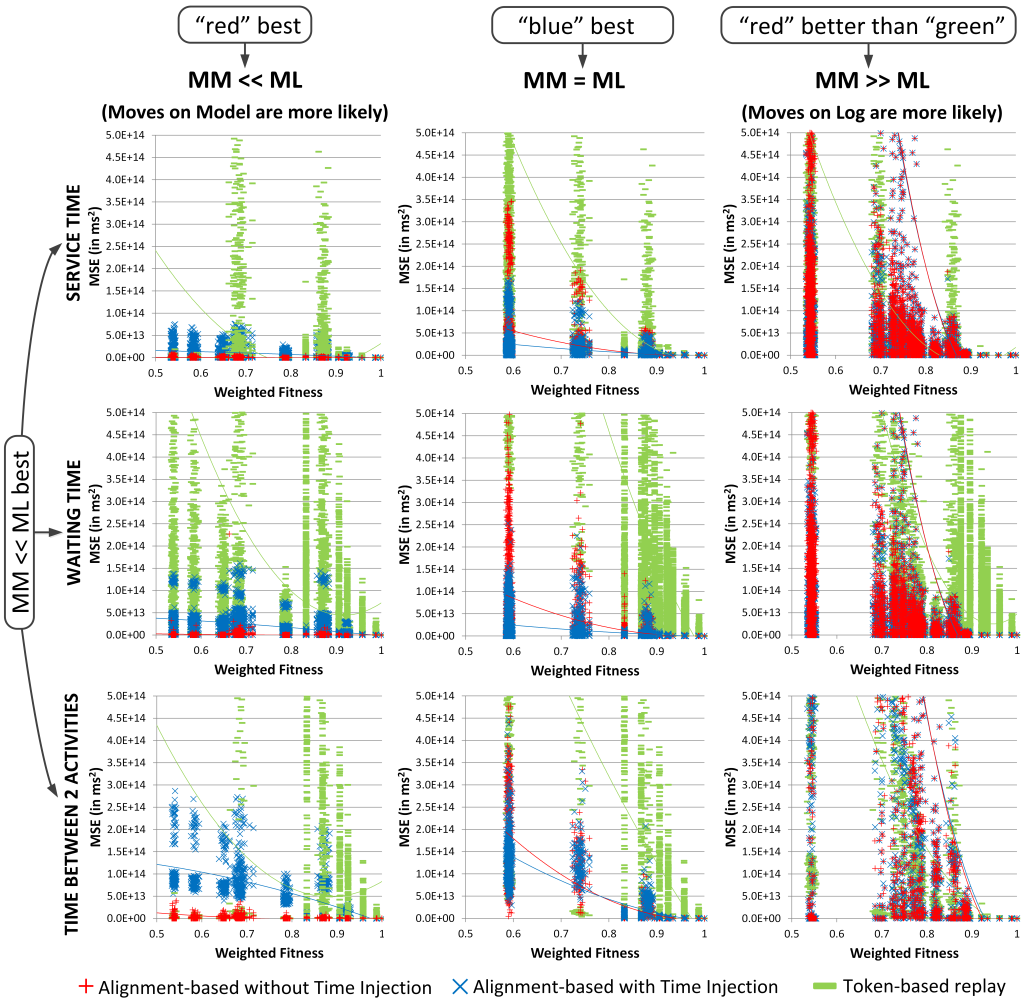


Figure 9.13: The Mean Square Errors (MSE) of performance measurements obtained using different alignment-based approaches and the token based approach [85], presented in the form of scatter plots. A mark in a scatter plot shows an MSE of a performance metric (i.e., average service/waiting time of an activity or the average time spent between two sequential activities), measured from a log of 1,000 traces. As shown in the figure, all approaches provide accurate measurements when the fitness values are perfect (1.00). The token-based replay approach returns high MSE values compared to the alignment-based approaches when measuring performance based on non-fitting logs. Furthermore, the alignment-based approach with likelihood cost function MM $\ll$ ML without timestamp injection provides the lowest MSE values compared to other approaches.

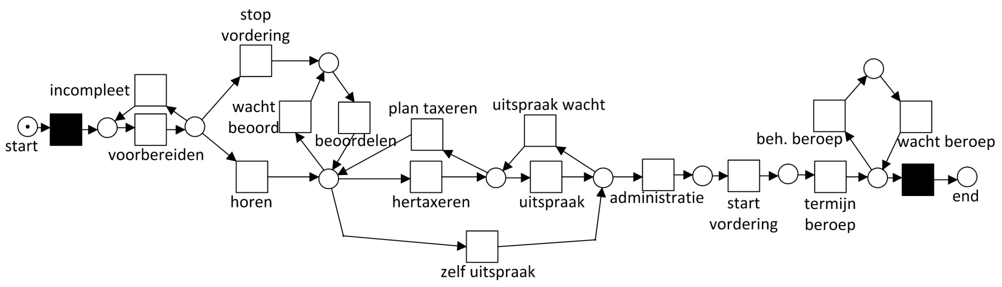


Figure 9.14: A property valuation objection handling in a Dutch municipality, given in terms of a Petri net.

cost function MM $\gg$ ML many moves on log are constructed instead of moves on model when a non fitting events occur. This implies that only few moves on model are constructed and only few timestamps injections were performed. From the scatter plots in Figure 9.13, we see that the alignment-based approach with cost function MM $\ll$ ML and without any timestamps injection yields lower MSEs compared to other approaches. Thus, this approach yields the most accurate performance measurements.

9.4.2 Case Study

To illustrate the applicability of the approach mentioned in Section 9.3, we took the "objection to property valuation handling process" of a municipality in the Netherlands as a case study. Every year, property owners in the Netherlands receive a new assessment of the value of their property. The value of a property forms the basis of the calculation of several municipality taxes, such as property tax, income tax, and water charges. If a property owner objects against an assessment to the value of his property, he can file an objection to the municipality where he is currently resided within six weeks since the moment the assessment was received. The net in Figure 9.14 models how such an objection is handled.

The filed objection form is first scanned by a legal officer (voorbereiden). If some additional documents are required then the form is marked as an incomplete form and sent back to the person who filed it (incomplete). If the form is completed but requires further explanations, the officer contacts the person who filed it to get an oral explanation (horen). Otherwise, the officer removes all taxes whose value are determined based on the property value (stop vordering) and then starts evaluating the objection thoroughly (beoordelen). The officer may delegate the evaluation to other officers if he/she does not have sufficient knowledge to perform the evaluation. Such a delegation is recorded explicitly as an instance of activity wacht beoord. After the objection is evaluated, the officer either issues a decision (zelf uitspraak) or asks an independent third party to re-estimate the property value (hertaxeren) before making a decision (uitspraak). The officer may ask the independent third party to re-estimate the value of the property more than once (plan taxeren). The officer may also undo the decision by performing activity uitspraak wacht. After a decision is legalized by an administration officer (administratie), it can not be undone. Based on the legalized decision, the administration officer updates the values of taxes that are influenced by the decision and notifies them to the person who file the objection. After the person gets a notification, the legal officer set up an account in an online system to allow the person to appeal against the decision within a period of time (termijn beroep). If the person appeals against the decision then an officer from the legal

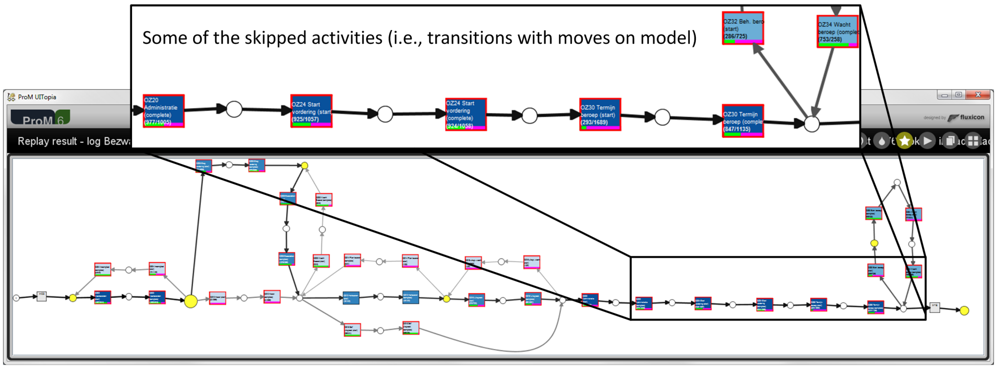
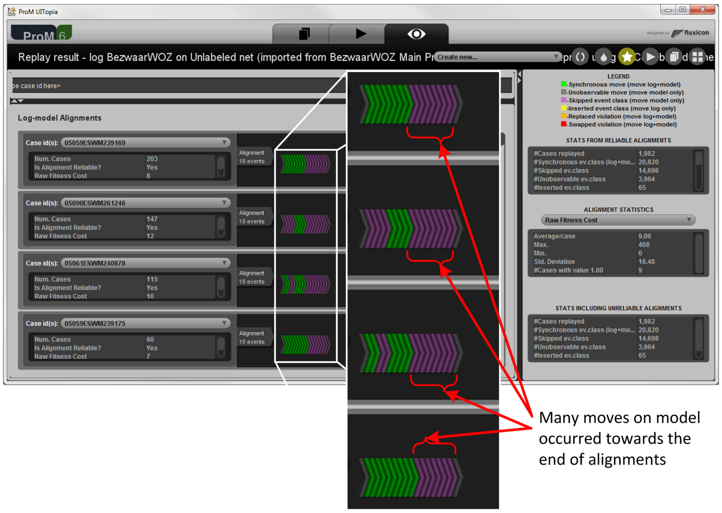

Some of the skipped activities (i.e., transitions with moves on model)
OZ20 Administratie (complete) (977/1005)
OZ24 Start vordering (start) (925/1057)
OZ24 Start vordering (complete) (924/1058)
OZ30 Termijn beroep (start) (293/1689)
OZ30 Termijn beroep (complete) (847/1135)
OZ32 Beh. beroep (start) (286/725)
OZ34 Wacht beroep (complete) (753/258)
ProM UITopia
ProM 6
Replay result - log Bezwa

Figure 9.15: Projection onto model of the constructed multi-set of alignments between the log and the low level net of Figure 9.14. Moves on model occurred in almost all transitions.



Replay result - log BezwaarWOZ on Unlabeled net (imported from BezwaarWOZ Main Pr

pe case id here>

Log-model Alignments

Case id(s): 05059ESWM239169

Num. Cases	203
Is Alignment Reliable?	Yes
Raw Fitness Cost	8

Alignment 18 events

Case id(s): 05090ESWM261246

Num. Cases	147
Is Alignment Reliable?	Yes
Raw Fitness Cost	12

Alignment 18 events

Case id(s): 05061ESWM240878

Num. Cases	115
Is Alignment Reliable?	Yes
Raw Fitness Cost	10

Alignment 18 events

Case id(s): 05059ESWM239175

Num. Cases	60
Is Alignment Reliable?	Yes
Raw Fitness Cost	7

Alignment 18 events

LEGEND
🟩-Synchronous move (move log+model)
⬜-Unobservable move (move model only)
🟪-Skipped event class (move model only)
🟨-Inserted event class (move log only)
🟧-Replaced violation (move log+model)
🟥-Swapped violation (move log+model)

STATS FROM RELIABLE ALIGNMENTS

#Cases replayed	1,982
#Synchronous ev.class (log+mo...	20,820
#Skipped ev.class	14,698
#Unobservable ev.class	3,964
#Inserted ev.class	65

ALIGNMENT STATISTICS

Raw Fitness Cost

Average/case	9.06
Max.	408
Min.	0
Std. Deviation	16.48
#Cases with value 1.00	9

STATS INCLUDING UNRELIABLE ALIGNMENTS

#Cases replayed	1,982
#Synchronous ev.class (log+mo...	20,820
#Skipped ev.class	14,698
#Unobservable ev.class	3,964
#Inserted ev.class	65

Figure 9.16: Projection onto log of the constructed multi-set of alignments between the log and the low level net of Figure 9.14. Many moves on model occurred towards the end of alignments.

department of the municipality is dispatched to handle the appeal (beh. beroep). After the appeal is handled, the officer set up the online system again to allow for another appeal (wacht beroep). If there are no appeals within a period of time, the online system automatically terminates the case.

The event log for the case study contains events that were collected by observing the execution of the process during period of 8.5 months. The log consists of 1,982 cases and 20,885 events where both start and completion of activity instances are recorded. We followed the steps previously described in Section 9.3 to measure performance. We constructed a low level net of the high level net in Figure 9.14 and then computed a multi-set of alignments from the log and the low level net. We use a basic and optimal oracle function with respect to a likelihood cost function where the costs of moves on model are much cheaper than the costs of moves on log, i.e., MM:ML = 1:50. The weighted fitness between the multi-set of alignments and the model is 0.72.

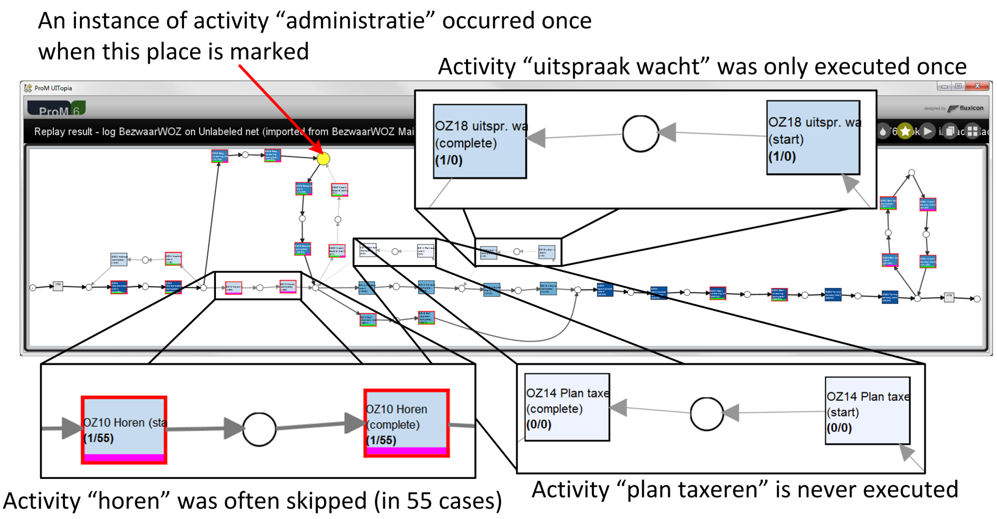

An instance of activity “administratie” occurred once when this place is marked
Activity “uitspraak wacht” was only executed once
Replay result - log BezwaarWOZ on Unlabeled net (imported from BezwaarWOZ Mai
OZ18 uitspr. wa (complete) (1/0)
OZ18 uitspr. wa (start) (1/0)
OZ10 Horen (sta (1/55)
OZ10 Horen (complete) (1/55)
OZ14 Plan taxe (complete) (0/0)
OZ14 Plan taxe (start) (0/0)
Activity “horen” was often skipped (in 55 cases)
Activity “plan taxeren” is never executed

Figure 9.17: Projection onto model of the constructed multi-set of alignments between the filtered log and the low level net of Figure 9.14.

Figure 9.15 shows the constructed multi-set of alignments, projected onto the low level net. As shown in the figure, deviations of type “move on model” occur in almost all transitions. Figure 9.16 shows some alignments in the multi-set that occur frequently. As shown in Figure 9.16, often several moves on model can be observed towards the alignment. This indicates that many cases in the log are incomplete.

Therefore, we filtered out cases that are not yet completed. According to the net in Figure 9.12, a properly completed case should contain an instance of activity termijn beroep. Therefore, we filtered out cases whose alignments do not contain any synchronous move of activity termijn beroep. We obtained an event log consisting of 293 cases with 4,124 events. We constructed alignments for each trace in the filtered log and the low level net of Figure 9.12. The weighted fitness of the multi-set is 0.84. Note that this value is higher than the weighted fitness obtained for the multi-set of alignments of the original log. The projection onto model of the multi-set is shown in Figure 9.17. Figure 9.17 shows that activity uitspraak wacht rarely occurred. No instance of activity plan taxeren is observed in the filtered log. These may indicate that both uitspraak wacht and plan taxeren are activities that are only performed in exceptional cases. The figure also shows that activity horen is often skipped. In addition, the figure also indicates less number of moves on log compared to the one shown in Figure 9.15.

Based on the experimental results in Section 9.4.1, performance measurements obtained from a log and model with a weighted fitness of 0.84 using alignment-based approach with likelihood cost function MM $\ll$ ML and without timestamps injections are relatively accurate. Thus, we used this approach to compute performance values. Figure 9.17 shows the results of these measurements, projected onto the low level net. As shown in the figure, the performance bottleneck of the process lies in the time spent to wait for the execution of activity uitspraak. In average, it takes 4.77$\pm$0.80 months (i.e., 0.80 is the 95% confidence interval) from the moment all non human resources required to perform activity uitspraak are available until the moment a human resource execute the activity. The average case throughput time , measured from the moment

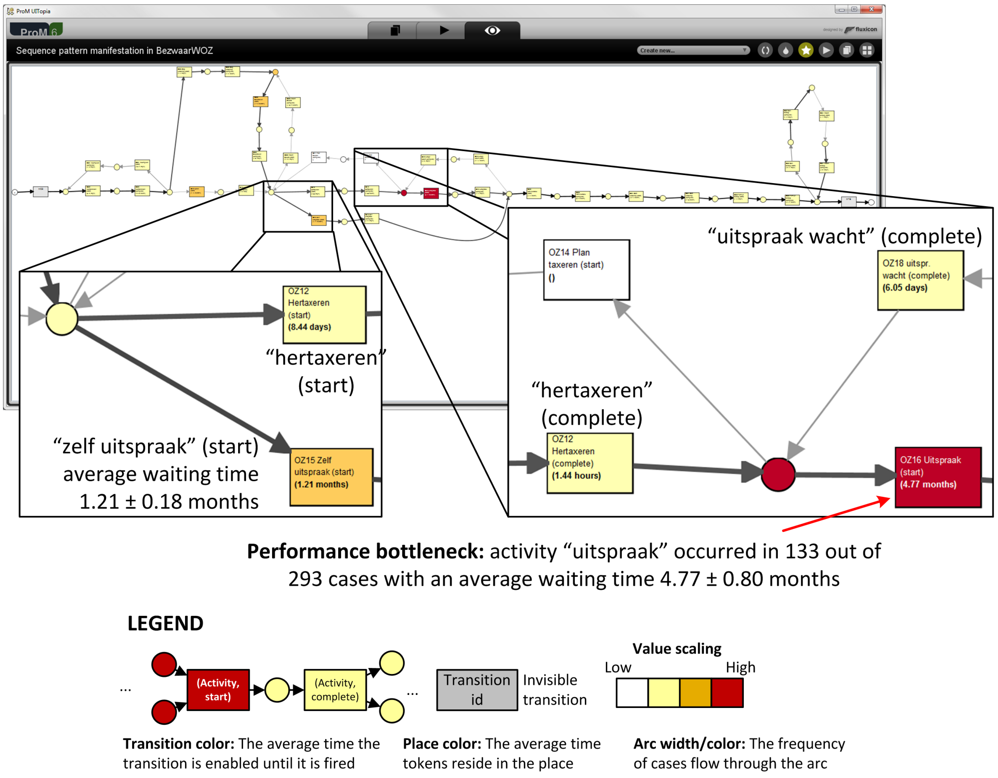

ProM 6
Sequence pattern manifestation in BezwaarWOZ
OZ12 Hertaxeren (start) (8.44 days)
"hertaxeren" (start)
"zelf uitspraak" (start) average waiting time 1.21 ± 0.18 months
OZ15 Zelf uitspraak (start) (1.21 months)
OZ14 Plan taxeren (start) ()
"uitspraak wacht" (complete)
OZ18 uitspr. wacht (complete) (6.05 days)
"hertaxeren" (complete)
OZ12 Hertaxeren (complete) (1.44 hours)
OZ16 Uitspraak (start) (4.77 months)
Performance bottleneck: activity "uitspraak" occurred in 133 out of 293 cases with an average waiting time 4.77 ± 0.80 months
LEGEND
(Activity, start)
(Activity, complete)
Transition id
Invisible transition
Value scaling
Low
High
Transition color: The average time the transition is enabled until it is fired
Place color: The average time tokens reside in the place
Arc width/color: The frequency of cases flow through the arc

Figure 9.18: The ProM 6 screenshot, showing the performance measurements obtained from the real-life log projected onto the low level net. Performance bottleneck of the process is shown by the dark-colored transitions/places.

each case in the log is started until the moment the last event in the case occurred, is 5.16±0.59 months. This means that the average waiting time for uitspraak is more than half of the average time spent to finish a case. The average time spent between the moment an instance of hertaxeren is completed until activity uitspraak is started in all cases is 5.05±0.04 months. There is only one case where an instance of activity uitspraak was directly preceded by an instance of uitspraak wacht. The time spent between the completion of uitspraak wacht and the start of uitspraak for the case is only 6.06 days. Therefore, adding some resources that can perform uitspraak after activity hertaxeren may improve the performance of overall process significantly.

Activity uitspraak is only executed if activity hertaxeren is chosen over zelf uitspraak. Figure 9.17 shows that the average time spent waiting a resource to perform activity zelf uitspraak (1.21±0.18 months) is much less than the average time spent waiting a resource to perform activity uitspraak (i.e., 4.77±0.80 months). Thus, increasing the number of cases that perform activity zelf uitspraak instead of hertaxeren may improve the overall performance of the process.

Figure 9.19 shows the average service time of activities measured from the constructed activity instances. Vertical lines show 95% confidence interval value of the measured service time. The average time spent to perform activities administratie, hertaxeren, horen, start vordering, stop vordering, and wacht beoord are relatively low (below

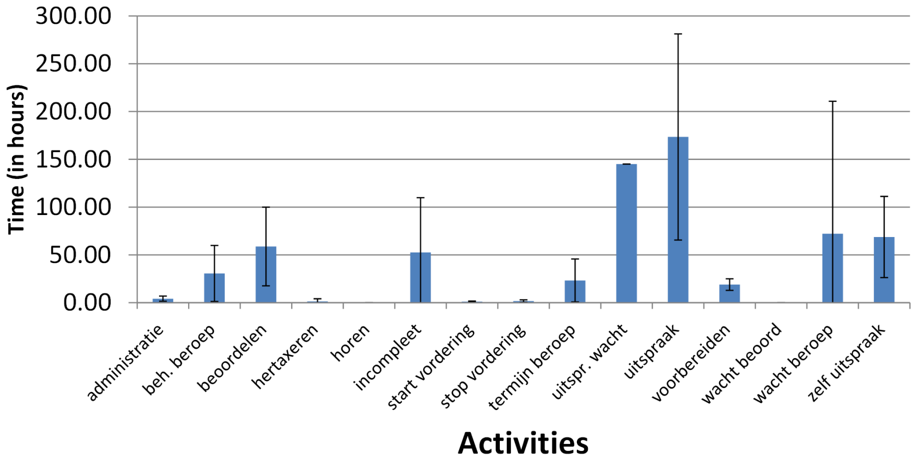


Figure 9.19: The average service time for each activity (including 95% confidence interval), obtained from the filtered log.

24 hours). Furthermore, we also see that the average service time of both uitspraak and uitspraak wacht are relatively high compared to the service time of other activities (above 100 hours). This may be an indication that not enough resources are allocated/allowed to perform the two activities. Improving the service time of the two activities may improve the overall process significantly.

9.5 Conclusion

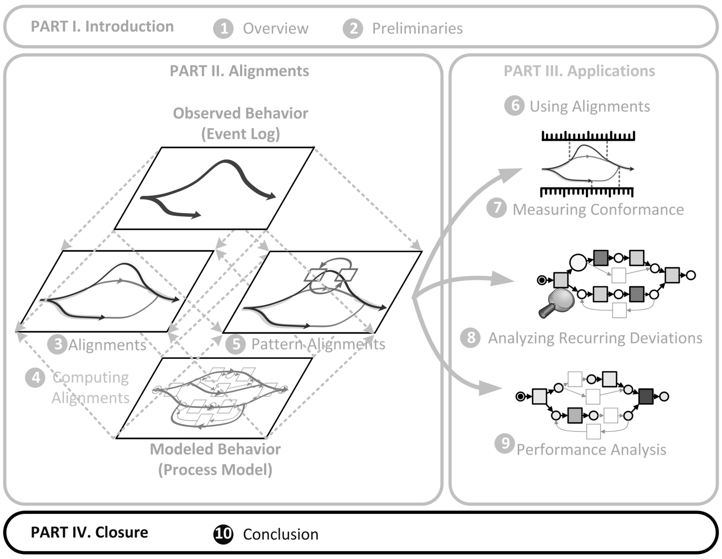
Process performance measurement is a crucial step to improve the overall performance of organizations. However, in an environment where process executions may deviate from prescribed process models, measuring performance is not a trivial task. Given a complex event log and a high level Petri net, we presented an alignment-based approach to measure performance accurately even if the log is not perfectly fitting the net. We showed that the approach is better than the token based replay approach when dealing with non fitting logs. We also showed that the accuracy of performance measurements can be estimated from the weighted fitness value between the log and the low level net of the original net. Moreover, we showed that the approach typically provides the most accurate performance measurements using a likelihood cost function that assigns higher costs to moves on model (compared to moves on log). Note that ultimately, the accuracy of the approach depends on the quality of the event log, i.e., the completeness of the events in the log required to measure performance. A perfectly fitting log does not guarantee accurate performance measurements if it only contains a small set of real process executions, i.e., many cases are needed to ensure reliable estimates.

Part IV

Closure

Chapter 10

Conclusion



In this chapter, we summarize our main findings. In Section 10.1, we highlight the results and contributions of this thesis. Section 10.2 lists some of the remaining challenges/open questions related to the techniques proposed in this thesis. Finally, Section 10.3 provides some possible directions for further research.

10.1 Contributions of This Thesis

The main theme of this thesis is the relationship between process models and event data. We showed that in the presence of observed behavior and modeled behavior, *aligning them is crucial for various types of process analysis*. In Chapter 3, we formalized the notion of alignments between the observed behavior in an event log and the modeled behavior in a process model. An alignment between a trace in an event log and a model is a pairwise comparison between executed activities in the trace and the activities allowed by the model. If the trace is not perfectly fitting the model, the alignment may contain a pair of activities with a “no move” represented as “$\gg$”. The model may not be able to mimic an event in the log or vice versa. The “no moves” in the alignment explicitly show what deviations occur and where the deviations are.

In this thesis, we chose Petri nets as a process modeling formalism due to their expressiveness, clarity, and simple graphical notation. However, the results are applicable to other process modeling languages for which translations into Petri nets that preserve the set of allowed traces are available (e.g., BPMN, BPEL, YAWL).

We showed that our notion of alignments can be implemented and used in practice. In Chapter 4, we showed a memory-efficient approach to compute alignments, given a trace and a Petri net. We demonstrated that the problem of computing optimal alignments can be modeled as the problem of computing a shortest path between two nodes in a directed graph. Thus, we can use existing shortest-path algorithms like the $\mathbb{A}^{\star}$ algorithm to compute an optimal alignment between them. We showed that the approach is extendable to Petri nets with reset/inhibitor arcs. Although constructing alignments is computationally expensive, we showed that the approach works well in both experimental and real-life settings. We also introduced some variants of alignments, such as the representative alignments and prefix alignments. We showed that the variants can be efficiently computed in a similar way as computing one optimal alignment per trace.

In Chapter 5, we showed that the approach to compute alignments can be used to identify high-level deviations such as swapped and replaced activities. Given a trace and a Petri net, we showed examples of high-level deviation patterns that can be instantiated to identify high-level deviations in the trace. Furthermore, we also showed that the approach can be used to identify possible root causes of deviations, taking into account both high-level and low-level deviations between the trace and the net.

Once alignments between the observed behavior in an event log and the modeled behavior in a process model are obtained, we showed that many types of analysis can be performed in a robust way. Different applications of alignments were highlighted in Part III of this thesis.

Given an event log and a Petri net, in Chapter 7 we showed that various alignment-based metrics to measure conformance between the log and the net yield intuitive results even if the log is non-fitting. We showed that the alignment-based fitness metric is robust to possible peculiarities of Petri nets, such as duplicate/invisible transitions and complex control-flow patterns. Furthermore, we showed that the alignment-based precision metrics are unidimensional and more robust to non-fitting logs compared to existing metrics.

In Chapter 8, we showed that alignments can be used to analyze recurring deviations. Three alignment-based visualizations to highlight recurring deviations between traces of an event log and a Petri net were proposed: (1) projection of alignments onto process models, (2) projection of alignments onto event logs, and (3) aligned alignments. Each visualization highlights a specific aspect of deviations and therefore they complement each other. Using a case study, we showed how the visualizations, in combination with data mining techniques, can be used to identify possible root causes of deviations and recurring deviations. Moreover, we also showed how the alignments can be used to repair process models to better reflect the observed reality in event logs.

Finally, in Chapter 9 we showed that alignments are crucial to measure performance metrics accurately for a given event log and process model. Experimental evaluations showed that alignment-based performance measurements deal better with non-fitting logs than the current state-of-the-art techniques in literature. We showed that even if the log is not fitting, process performance can be measured accurately. Using a case study, we showed that the technique provides useful insights into performance bottlenecks.

All approaches mentioned in this thesis are implemented in the ProM 6.3 framework under the following packages: PNetReplayer, PNetAlignmentAnalysis, and ETConformance.[1] All packages are publicly available from `http://www.processmining.org`.

10.2 Challenges and Open Issues

The notion of alignments is crucial for many types of analysis based on a given event log and process model. We list some challenges and open issues related to the techniques presented in this thesis:

- **Constructing likelihood cost functions**. Most of the experiments in this thesis use the standard likelihood cost function that penalizes all moves on log and moves on model equally (except the moves on model of invisible transitions). In practice, these movements may have a different likelihoods. Given an event log and a process model, determining the "ideal" likelihood cost of movements for logged activities and tasks in the model remains an open issue. In an ideal situation, such a likelihood cost is determined by an expert, but this is not always trivial. One idea to do this automatically is to exploit the information that exist in the log and in the model, such as the frequency of activities (frequent/infrequent activities have less cost) or the possible behavior after performing an activity according to the model. If more information is available, we may also consider the semantics of activities. For example in a claim handling process of an insurance company, paying a claimant multiple times (i.e., moves on log) may have high likelihood cost (i.e., it is less likely occur). Further research is needed to investigate the extent such likelihood cost functions can be automatically derived.
- **Alignments in the large**. Given a trace, a Petri net, and a likelihood cost function, the approach to compute an optimal alignment between the trace and the net in this thesis is shown to be efficient under an assumption that they are both fit in a computer memory (see Chapter 4). However, this assumption may not always be satisfied. For example, in the healthcare domain, the number of medical devices that log their activities is steadily increasing. The massive amount of data produced by such medical devices can be challenging even for the efficient approach proposed in this thesis. Many approaches to analyze big data typically use some heuristics to deal with the complexity of the data (e.g. [19]). However, applying such heuristics to compute alignments may yield misleading results. Several approaches to partition the net and later distribute the computation of alignments were proposed recently (e.g., [117, 118, 172, 173, 199]). Such approaches are expected to be scalable and require lower computation time in dealing with big data. Another possible solution is to exploit concurrency of transitions, similar to the way stubborn sets are exploited for various Petri net analysis [93, 154, 155]. It is also interesting to investigate model-checking approaches such as the sweep-line method [35, 87] to delete hopeless states and thus decrease the memory usage of the proposed techniques in this thesis.
- **Computing all alignments between a trace and a process model**. In Section 4.6, we provided an approach to compute all optimal alignments between a given trace and Petri net. However, experiments show that the approach is limited to logs and models with moderate complexity because it requires an exhaustive exploration of

[1]The ETConformance package is maintained by Jorge Munoz, Universitat Politecnica de Catalunya (UPC).

a large state space that needs to fit in memory. To decrease the memory requirement, we may be able to use existing data compression techniques, exploit concurrencies between activities, or use a special data structure to store alignments in a compact way.

- **Online alignments**. In Section 4.5, we proposed an approach to compute a prefix optimal alignment from a given trace, a process net, and a likelihood cost function. Such an alignment can be used to check deviations in an online setting where the trace may not be completed. To compute the alignment, we used an underestimation function that always returns the value of 0. As shown in Section 4.7, such a naive underestimation function yields a high computation time and a low memory-efficiency for computing optimal alignments. In an online setting, it is often desirable to have results with low computation time. One of the possibilities to improve both computation time and memory-efficiency of the approach is to use a similar underestimation function as the one used in Section 4.4.2 with inequalities for places of process nets. This way, not only do we improve the computation time, but we can also handle Petri nets without the option to complete to a certain extent. However, experimental evaluations are needed to see whether the computation time gained from the precise measurement is less than the extra overhead time required to compute ILPs.

  Given an uncompleted trace, a Petri net, and a likelihood cost function, an optimal prefix alignment between the trace and the net may be totally different than an optimal prefix alignment of the same trace after appended with a new activity. This means that in an online setting, the diagnostics obtained from a prefix alignment between a trace and a Petri net may change after a new event for the trace occurs. For deviation analysis purposes, it is often desirable to have diagnostics that do not change after new events are appended (i.e., consistent diagnostics). One way to do this is to also consider sub-optimal prefix alignments for online diagnostics purpose and exploit historical information to determine which alignments are most likely followed by an uncompleted trace.

- **Log alignments**. Given an event log and a process model, alignments are computed for each trace in the log. We assume that each trace in the log is independent from all other traces in the same log. In practice, this assumption may not always hold. For example, an employee of an insurance company may be involved in multiple claims at a time. Thus, the execution of a trace may indirectly influence the execution of other traces in the same log. Instead of considering each trace independently from other traces in the same log, one can compute alignments by considering all traces in the log at once. One possible solution is to extend the approach presented in Chapter 4 by creating an event net for each trace in the log, modeling dependencies between traces, and then performing the same approach presented in the chapter. This way, we can compute a set of alignments with the minimum total cost. However, it is easy to see that the computational complexity of such an approach is extremely high.

10.3 Future Research

In this thesis, we only showed three possible applications of alignments: (1) measuring conformance, (2) identifying possible root causes of deviations, and (3) measuring performance. However, there are potentially many other applications of alignments. In the remainder, we sketch some research directions that further exploit the notion of alignments.

- **Beyond control-flow alignments**. Except for the discussion in Section 6.4, the notion of alignments in this thesis only takes into account the control-flow perspective of process models. In practice, some process modeling languages such as BPMN can also be used to model data (e.g., resources that are allowed to perform some tasks and conditions that need to be satisfied to perform an activity). Aligning a trace and a process model by taking into account extra perspectives other than control-flow is a challenging task that has many potential uses. Constructing control-flow aware alignments is already a computationally challenging task. Adding another dimension to the alignment computation increases the complexity of the approach manifold. Thus, approaches to replay event logs to Petri nets with data [44, 45] use heuristics to tackle such problems. However, they do not guarantee the "optimality" of the obtained alignments with respect to all considered dimensions.
- **Process model repair**. In Chapter 8, given an event log and a process model, we showed that the alignments between each trace of the log and the model show insights that can be exploited to repair the original process model (i.e., construct a model with better fitness and precision than the original model with respect to the log). The current relies on the user to perform iterative improvements on the original process model. Fahland and Van der Aalst [62] proposed an automated approach to construct a model that is similar as the original model, but reflects the behavior observed in the log better. However, the approach only considers the fitness perspective. In [24], Buijs et al. proposed an approach to repair process models based on the genetic algorithms. The approach considers all conformance dimensions at once to separate good models and bad models. However, genetic algorithms often require long computation times before good results are obtained. A fully automated approach to repair process models taking into account all dimensions of conformance with less complexity than genetic algorithms is an interesting topic for further research.
- **Deviation-aware concept drift analysis**. In the area of process mining, many analysis techniques based on both event logs and process models assume that the logs reflect the behavior of their processes in a steady state. However, experience shows that this assumption does not always hold. Processes may change while they are being analyzed. For example, hospitals and insurance companies may change their operational procedures in emergency situations. When a major disaster occurs, insurance companies may skip investigations for all claims below certain amount of money. Existing process mining approaches to deal with concept drift mainly focus on: (1) detecting change points, and (2) characterizing and localizing changes [22]. To detect change points, existing approaches (e.g., [22, 30]) rely on the feature of event logs without considering process models. In the presence of both a process model and an event log, alignments provide a way to also include deviations to predefined process models as an extra feature in the change points analysis. This inclusion of alignments for concept drift analysis may improve the quality of the existing approaches.
- **Alignment-based operational support**. Application examples of alignments provided in this thesis are restricted to *offline analysis*. Given an incomplete trace and a process model, a prefix optimal alignment between the trace and the model does

not only provide diagnostics of occurring deviations, but also information about the current state of the trace according to the model. This information can be exploited to provide operational support (e.g., online prediction, deviation analysis, and recommendations). For example, given an event log and a process model, we construct a multi-set of alignments from the log and the model. Then, we build a predictor based on the alignment multi-set for each visited state in the model. The set of predictors are then used to provide recommendations. A similar idea was already proposed in [37, 150, 183], but all of them require the log to be perfectly fitting. It is an interesting research question whether the notion of alignments can be used to remove this requirement without sacrificing the quality of operational supports.

The concept of alignments in this thesis is not bound to any specific domain. It is however interesting to investigate how alignments can be tailored to solve problems in specific domains. In the healthcare domain, process models are often used to guide process executions, but the flexibility to deviate from predefined model (e.g., guidelines) is essential. For example, each patient in a hospital typically follows a unique treatment although there are certain procedures that describe how patients should be treated in general. Therefore, deviations to predefined process models often occur in the healthcare domain. In contrast, processes in municipalities are typically less flexible. Deviations to predefined process models are less likely occur in a municipalities than in a healthcare domain. These characteristics influence the types of analysis that are suitable for each domain. In the healthcare domain, identifying reoccurring deviations may be more useful than identifying the average throughput time of all cases. In contrast, the information such as the average throughput time of all cases can be more beneficial for municipalities than the information about recurring deviations.

The relation between observed behavior and modeled behavior can be used in many different domains. Case studies in the area of healthcare [20, 102, 140], telecommunications [67], finance [3], security [11, 12, 16, 179], auditing [23, 86, 188], and manufacturing [146, 147] illustrate the broad spectrum of applications that ma benefit from the results in this thesis. Moreover, the work in this thesis is also crucial in the process mining domain, as it provides a solid basis to evaluate process discovery approaches [25, 148] and enables various types of process model enhancements based on event logs.

Bibliography

[1] 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.

[2] A. Adriansyah. Performance Analysis of Business Processes from Event Logs and Given Process Models. Master's thesis, Eindhoven University of Technology, 2009.

[3] A. Adriansyah and J.C.A.M. Buijs. Mining Process Performance from Event Logs. In M. La Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 217–218. Springer Berlin Heidelberg, 2013.

[4] A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment Based Precision Checking. In M. La Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 137–149. Springer Berlin Heidelberg, 2013.

[5] A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Measuring Precision of Modeled Behavior. *Information Systems and e-Business Management*, pages 1–31, 2014.

[6] A. Adriansyah, N. Sidorova, and B.F. van Dongen. Cost-Based Fitness in Conformance Checking. *International Conference on Application of Concurrency to System Design*, pages 57–66, 2011.

[7] A. Adriansyah, B.F. van Dongen, D. Piessens, M.T. Wynn, and M. Adams. Robust Performance Analysis on YAWL Process Models with Advanced Constructs. *Journal of Information Technology Theory and Application (JITTA)*, 12(3):5–26, 2011.

[8] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, EDOC '11, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society.

[9] 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 Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer Berlin Heidelberg, 2011.

[10] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Memory-Efficient Alignment of Observed and Modeled Behavior. BPM Center Report BPM-03-03, BPMcenter.org, 2013.

[11] A. Adriansyah, B.F. van Dongen, and N. Zannone. Controlling Break-The-Glass Through Alignment. *ASE SCIENCE*, 2(4):198–212, 2013.

[12] A. Adriansyah, B.F. van Dongen, and N. Zannone. Privacy Analysis of User Behavior Using Alignments. *it – Information Technology*, 55(6):255–260, 2013.

[13] R. Agrawal, T. Imieliński, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Record*, 22(2):207–216, June 1993.

[14] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[15] R. Alur and T.A. Henzinger. A Really Temporal Logic. *Journal of the ACM (JACM)*, 41(1):181–203, January 1994.

[16] S. Banescu and N. Zannone. Measuring Privacy Compliance with Process Specifications. In *Proceedings of the 2011 Third International Workshop on Security Measurements and Metrics*, METRISEC '11, pages 41–50, Washington, DC, USA, 2011. IEEE Computer Society.

[17] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust Management System. In *Proceedings of the Second International Conference on Financial Cryptography*, FC '98, pages 254–274, London, UK, UK, 1998. Springer-Verlag.

[18] C.G.E. Boender and A.H.G.R. Kan. A Bayesian Analysis of the Number of Cells of a Multinomial Distribution. *Journal of the Royal Statistical Society*, 32(1-2):240–248, 1983.

[19] R.P.J.C Bose. *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*. PhD thesis, Eindhoven University of Technology, 2012.

[20] R.P.J.C. Bose and W.M.P. van der Aalst. Analysis of Patient Treatment Procedures. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 165–166. Springer Berlin Heidelberg, 2012.

[21] R.P.J.C. Bose and W.M.P. van der Aalst. Process Diagnostics using Trace Alignment: Opportunities, Issues, and Challenges. *Information Systems*, 37(2):117–141, April 2012.

[22] R.P.J.C. Bose, W.M.P. van der Aalst, I. Zliobaite, and M. Pechenizkiy. Handling Concept Drift in Process Mining. In H. Mouratidis and C. Rolland, editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer Berlin Heidelberg, 2011.

[23] M. Bozkaya, J. Gabriels, and J.M.E.M. van der Werf. Process Diagnostics: A Method Based on Process Mining. In *Information, Process, and Knowledge Management, 2009. eKNOW '09. International Conference on*, pages 22 –27, feb. 2009.

[24] J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. van Dongen, and W.M.P. van der Aalst. Improving Business Process Models using Observed Behavior. In P. Cudre-Mauroux, P. Ceravolo, and D. Gasevic, editors, *Proceedings of 3rd International Symposium on Data-driven Process Discovery and Analysis*, pages 44–59. Springer, Campione d'Italia, Italy, 2013.

[25] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In R. Meersman, H. Panetto, T. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I.F. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2012.

[26] 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 Their 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 Berlin Heidelberg, 2012.

[27] G. Canfora, F. García, M. Piattini, F. Ruiz, and C.A. Visaggio. A Family of Experiments to Validate Metrics for Software Process Models. *Journal of Systems and Software*, 77(2):113–129, August 2005.

[28] J. Cardoso. Control-flow Complexity Measurement of Processes and Weyuker's Properties. *Enformatika*, 8:213–218, October 2005.

[29] J. Cardoso, J. Mendling, G. Neumann, and H.A. Reijers. A Discourse on Complexity of Process Models. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 117–128. Springer Berlin Heidelberg, 2006.

[30] J. Carmona and R. Gavaldà. Online Techniques for Dealing with Concept Drift in Process Mining. In J. Hollmèn, F. Klawonn, and A. Tucker, editors, *IDA*, volume 7619 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2012.

[31] M. Castellanos, F. Casati, M. Shan, and U. Dayal. iBOM: A Platform for Intelligent Business Operation Management. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 1084–1095, Washington, DC, USA, 2005. IEEE Computer Society.

[32] F. Chesani, M. Gavanelli, M. Alberti, E. Lamma, P. Mello, and P. Torroni. Specification and Verification of Agent Interaction using Abductive Reasoning. In *Proceedings of the 6th international conference on Computational Logic in Multi-Agent Systems*, CLIMA'05, pages 243–264, Berlin, Heidelberg, 2006. Springer-Verlag.

[33] F. Chesani, P. Mello, M. Montali, and S. Storari. Testing Careflow Process Execution Conformance by Translating a Graphical Language to Computational Logic. In *Proceedings of the 11th conference on Artificial Intelligence in Medicine*, AIME '07, pages 479–488, Berlin, Heidelberg, 2007. Springer-Verlag.

[34] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2013.

[35] S. Christensen, L.M. Kristensen, and T. Mailund. "a sweep-line method for state space exploration". In Tiziana Margaria and Wang Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer Berlin Heidelberg, 2001.

[36] F. Chu and X. Xie. Deadlock Analysis of Petri nets using Siphons and Mathematical Programming. *Robotics and Automation, IEEE Transactions on*, 13(6):793–804, 1997.

[37] R. Conforti, M. de Leoni, M. La Rosa, and W.M.P. van der Aalst. Supporting Risk-Informed Decisions during Business Process Execution. In C. Salinesi, M.C. Norrie, and O. Pastor, editors, *Advanced Information Systems Engineering*, volume 7908 of *Lecture Notes in Computer Science*, pages 116–132. Springer Berlin Heidelberg, 2013.

[38] J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pages 332 –341, 2001.

[39] J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8:147–176, April 1999.

[40] 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.

[41] C. Costello and O. Molloy. Building a Process Performance Model for Business Activity Monitoring. In W. Wojtkowski, G. Wojtkowski, M. Lang, K. Conboy, and C. Barry, editors, *Information Systems Development*, pages 237–248. Springer US, 2009.

[42] G.B. Dantzig and M.N. Thapa. *Linear Programming 1: Introduction*. Springer, 1997.

[43] M. de Leoni, F.M. Maggi, and W.M.P. van der Aalst. Aligning Event Logs and Declarative Process Models for Conformance Checking. In A. Barros, A. Gal, and E. Kindler, editors, *Proceedings of the 10th International Conference on Business Process Management, BPM'12*, volume 7481 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin Heidelberg, 2012.

[44] M. de Leoni and W.M.P. van der Aalst. Aligning Event Logs and Process Models for Multi-perspective Conformance Checking: An Approach Based on Integer Linear Programming. In F. Daniel, J. Wang, and B. Weber, editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 113–129. Springer Berlin Heidelberg, 2013.

[45] M. de Leoni, W.M.P. van der Aalst, and B.F. van Dongen. Data- and Resource-Aware Conformance Checking of Business Processes. In W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas, editors, *Business Information Systems*, volume 117 of *Lecture Notes in Business Information Processing*, pages 48–59. Springer Berlin Heidelberg, 2012.

[46] A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.

[47] A.K. Alves 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*, 14:245–304, 2007.

[48] J. de Weerdt, M. de Backer, J. Vanthienen, and B. Baesens. A Critical Evaluation Study of Model-log Metrics in Process Discovery. In M. zur Muehlen and J. Su, editors, *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 158–169. Springer Berlin Heidelberg, 2011.

[49] J. de Weerdt, M. de Backer, J. Vanthienen, and B. Baesens. A Robust F-measure for Evaluating Discovered Process Models. In *IEEE Symposium Series on Computational Intelligence*, pages 148–155. IEEE, 2011.

[50] R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[51] J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer Berlin Heidelberg, 2001.

[52] A. del Río-Ortega, M. Resinas, C. Cabanillas, and A. Ruiz-Cortés. On the Definition and Design-time Analysis of Process Performance Indicators. *Information Systems*, 38(4):470 – 490, 2013.

[53] B. Depaire, J. Swinnen, M. Jans, and K. Vanhoof. A Process Deviation Analysis Framework. In M. La Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 701–706. Springer Berlin Heidelberg, 2013.

[54] J. Desel. Validation of Process Models by Construction of Process Nets. In W.M.P van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 110–128. Springer Berlin Heidelberg, 2000.

[55] J. Desel and J. Esparza. *Free choice Petri nets*. Cambridge University Press, New York, NY, USA, 1995.

[56] J. Desel and W. Reisig. The Synthesis Problem of Petri nets. *Acta Informatica*, 33(4):297–315, 1996.

[57] R.M Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology*, 50(12):1281–1294, November 2008.

[58] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification, 2nd Edition*. Wiley, 2000.

[59] M. Dumas, M. La Rosa, J. Mendling, and H.A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

[60] M. Dumas, W.M.P. van der Aalst, and A.H. ter Hofstede, editors. *Process-Aware Information Systems : Bridging People and Software through Process Technology*. Wiley-Interscience, Hoboken, NJ, 2005.

[61] A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. *Acta Informatica*, 27(4):315–342, 1990.

[62] D. Fahland and W.M.P. van der Aalst. Repairing Process Models to Reflect Reality. In *Proceedings of the 10th international conference on Business Process Management*, BPM'12, pages 229–245, Berlin, Heidelberg, 2012. Springer-Verlag.

[63] D.R. Ferreira, F. Szimanski, and C.G. Ralha. A Hierarchical Markov Model to Understand the Behaviour of Agents in Business Processes. In M. La Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 150–161. Springer Berlin Heidelberg, 2013.

[64] Jr. G.D. Forney. The Viterbi Algorithm. *Proceedings of the IEEE*, 61(3):268 – 278, march 1973.

[65] K. Gerke, J. Cardoso, and A. Claus. Measuring the Compliance of Processes with Reference Models. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part I*, OTM '09, pages 76–93, Berlin, Heidelberg, 2009. Springer-Verlag.

[66] C. Giblin, A.Y. Liu, S. Müller, B. Pfitzmann, and X. Zhou. Regulations Expressed As Logical Models (REALM). In *Proceedings of the 2005 conference on Legal Knowledge and Information Systems: JURIX 2005: The Eighteenth Annual Conference*, pages 37–48, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[67] S. Goedertier, J. de Weerdt, D. Martens, J. Vanthienen, and B. Baesens. Process Discovery in Event Logs: An Application in the Telecom Industry. *Applied Soft Computing*, 11(2):1697–1710, 3 2011.

[68] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *The Journal of Machine Learning Research*, 10:1305–1340, 2009.

[69] G. Governatori and Z. Milosevic. Dealing with Contract Violations: Formalism and Domain Specific Language. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, EDOC '05, pages 46–57, Washington, DC, USA, 2005. IEEE Computer Society.

[70] G. Governatori, Z. Milosevic, and S. Sadiq. Compliance Checking between Business Processes and Business Contracts. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, EDOC '06, pages 221–232, Washington, DC, USA, 2006. IEEE Computer Society.

[71] 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.

[72] D. Grigori, F. Casati, U. Dayal, and M. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 159–168, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[73] S. Guha, W.J. Kettinger, and J.T.C. Teng. Business Process Reengineering. *Information Systems Management*, 10(3):13–22, 1993.

[74] C.W. Gunther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2009.

[75] C.W. Gunther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 328–343, Berlin, Heidelberg, 2007. Springer-Verlag.

[76] M.H.T. Hack. *Decidability Questions for Petri Nets*. PhD thesis, MIT, 1976.

[77] C. Haisjackl, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber. Making Sense of Declarative Process Models: Common Strategies and Typical Pitfalls. In S. Nurcan, H.A. Proper, P. Soffer, J. Krogstie, R. Schmidt, T. Halpin, and I. Bider, editors, *Enterprise Business-Process and Information Systems Modeling*, volume 147 of *Lecture Notes in Business Information Processing*, pages 2–17. Springer Berlin Heidelberg, 2013.

[78] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[79] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[80] M. Han, T. Thiery, and X. Song. Managing Exceptions in the Medical Workflow Systems. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 741–750, New York, NY, USA, 2006. ACM.

[81] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs. *IEEE Trans. Syst. Sci. and Cybernetics*, SSC-4(2):100–107, 1968.

[82] T. T. Hildebrandt and R. R. Mukkamala. Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs. In K. Honda and A. Mycroft, editors, *Proceedings of the Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software*, EPTCS, pages 59–73, 2010.

[83] A.H.M. Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation*. Springer-Verlag, 2010.

[84] J. Hong, I. Mozetic, and R.S. Michalski. Incremental Learning of Attribute-Based Descriptions from Examples, The Method and User's Guide. Technical report, Department of Computer Science, University of Illinois, Urbana, 1986. Report ISG 86-5, UIUCDCS-F-86-949.

[85] P.T.G. Hornix. Performance Analysis of Business Processes through Process Mining. Master's thesis, Eindhoven University of Technology, 2007.

[86] M. Jans, B. Depaire, and K. Vanhoof. Does Process Mining Add to Internal Auditing? An Experience Report. In T. Halpin, S. Nurcan, J. Krogstie, P. Soffer, E. Proper, R. Schmidt, and I. Bider, editors, *Enterprise Business-Process and Information Systems Modeling*, volume 81 of *Lecture Notes in Business Information Processing*, pages 31–45. Springer Berlin Heidelberg, 2011.

[87] K. Jensen, L.M. Kristensen, and T. Mailund. The sweep-line state space exploration method. *Theoretical Computer Science*, 429:169179, 2012.

[88] G. Juhás, R. Lorenz, and J. Desel. Can I Execute My Scenario in Your Net? In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 289–308. Springer Berlin Heidelberg, 2005.

[89] G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1998.

[90] M. El Kharbili, A.K. Alves de Medeiros, S. Stein, and W.M.P. van der Aalst. Business Process Compliance Checking: Current State and Future Challenges. In Peter Loos, Markus Nüttgens, Klaus Turowski, and Dirk Werth, editors, *MobIS*, volume 141 of *LNI*, pages 107–113. GI, 2008.

[91] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, 2003.

[92] Knowit-Information Systems. eBay Study: How to Build Trust and Improve the Shopping Experience. `http://knowwpcarey.com/article.cfm?cid=25&aid=1171`, May 2012. last accessed on 20th January 2013.

[93] L.M. Kristensen, K. Schmidt, and A. Valmari. Question-guided Stubborn Set Methods for State Properties. *Formal Methods in System Design*, 29(3):215–251, November 2006.

[94] A. Lapadula, R. Pugliese, and F. Tiezzi. A Calculus for Orchestration of Web Services. In R. Nicola, editor, *Programming Languages and Systems*, volume 4421 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin Heidelberg, 2007.

[95] R. Laue and V. Gruhn. Complexity Metrics for business Process Models. In W. Abramowicz and H.C. Mayr, editors, *BIS*, volume 85 of *LNI*, pages 1–12. GI, 2006.

[96] P. Lawrence, editor. *Workflow Handbook*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[97] G. Lee and J.M. Yoon. An Empirical Study on the Complexity Metrics of Petri Nets. *Microelectronics Reliability*, 32(3):323 – 329, 1992.

[98] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966.

[99] 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, *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 109–121. Springer Berlin Heidelberg, 2011.

[100] R. Lorenz, G. Juhas, R. Bergenthum, J. Desel, and S. Mauser. Executability of Scenarios in Petri nets. *Theoretical Computer Science*, 410(1213):1190 – 1216, 2009.

[101] 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 S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer Berlin Heidelberg, 2011.

[102] R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, and P.J.M. Bakker. Application of Process Mining in Healthcare – A Case Study in a Dutch Hospital. In A. Fred, J. Filipe, and H. Gamboa, editors, *Biomedical Engineering Systems and Technologies*, volume 25 of *Communications in Computer and Information Science*, pages 425–438. Springer Berlin Heidelberg, 2009.

[103] 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. `http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation`, June 2011. Last accessed 20th Jan 2013.

[104] A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.

[105] A. Martens. Analyzing Web Service based Business Processes. In *Proceedings of the 8th international conference, held as part of the joint European Conference on Theory and Practice of Software conference on Fundamental Approaches to Software Engineering*, FASE'05, pages 19–33, Berlin, Heidelberg, 2005. Springer-Verlag.

[106] F. Melchert, R. Winter, and M. Klesse. Aligning Process Automation and Business Intelligence to Support Corporate Performance Management. In *AMCIS*, page 507. Association for Information Systems, 2004.

[107] J. Mendling. Metrics for Business Process Models. In *Metrics for Process Models*, volume 6 of *Lecture Notes in Business Information Processing*, pages 103–133. Springer Berlin Heidelberg, 2009.

[108] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models Based on Metrics. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer Berlin Heidelberg, 2007.

[109] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[110] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[111] M. Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[112] M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on Web*, 4(1):3:1–3:62, January 2010.

[113] S. Moore. Business Intelligence Ranked Top Technology Priority by CIOs for Fourth Year in a Row. `http://www.gartner.com/it/page.jsp?id=888412`, 2009. last accessed on 17th January 2013.

[114] M.Petkovic, D. Prandi, and N. Zannone. Purpose Control: Did You Process the Data for the Intended Purpose? In Willem Jonker and Milan Petkovic, editors, *Secure Data Management*, volume 6933 of *Lecture Notes in Computer Science*, pages 145–168. Springer, 2011.

[115] J. Munoz-Gama and J. Carmona. A Fresh Look at Precision in Process Conformances. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer Berlin Heidelberg, 2010.

[116] J. Munoz-Gama and J. Carmona. Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, April 11-15, 2011, Paris, France*, pages 184–191. IEEE, April 2011.

[117] J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Conformance Checking in the Large: Partitioning and Topology. In F. Daniel, J. Wang, and B. Weber, editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 130–145. Springer Berlin Heidelberg, 2013.

[118] J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Hierarchical Conformance Checking of Process Models Based on Event Logs. In J. Colom and J. Desel, editors, *Application and Theory of Petri Nets and Concurrency*, volume 7927 of *Lecture Notes in Computer Science*, pages 291–310. Springer Berlin Heidelberg, 2013.

[119] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, August 2002.

[120] J. Nakatumba and W.M.P. van der Aalst. Analyzing Resource Behavior Using Process Mining. In S. Rinderle-Ma, S. Sadiq, and F. Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 69–80. Springer Berlin Heidelberg, 2010.

[121] S. B. Needleman and C.D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[122] OMG. Business Process Model and Notation (BPMN) Version 2.0, 2011.

[123] P.S. Pande, R.P. Neuman, and R.R. Cavanagh. *The Six Sigma Way: How GE, Motorola, and Other Top Companies are Honing Their Performance*. McGraw-Hill, 4 2000.

[124] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag.

[125] H. Petra, H. Stefan, J. Stefan, N. Jens, S. Katrin, and T. Michael. A Comprehensive Approach to Flexibility in Workflow Management Systems. *SIGSOFT Software Engineering Notes*, 24:79–88, March 1999.

[126] C. Pettey. Gartner EXP Worldwide Survey of Nearly 1,600 CIOs Shows IT Budgets in 2010 to be at 2005 Levels. `http://www.gartner.com/it/page.jsp?id=1283413`, 2010. last accessed on 17th January 2013.

[127] C. Pettey and L. Goasduff. Gartner Executive Programs' Worldwide Survey of More Than 2,300 CIOs Shows Flat IT Budgets in 2012, but IT Organizations Must Deliver on Multiple Priorities. `http://www.gartner.com/it/page.jsp?id=1897514`, 2012. last accessed on 17th January 2013.

[128] C. Pettey and R. van der Meulen. Gartner Executive Program Survey of More Than 2,000 CIOs Shows Digital Technologies Are Top Priorities in 2013. `http://www.gartner.com/newsroom/id/2304615`, 2013. last accessed on 17th January 2013.

[129] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H.A. Reijers. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 383–394. Springer Berlin Heidelberg, 2012.

[130] S.S. Pinter and M. Golani. Discovering Workflow Models from Activities Lifespans. *Computers in Industry*, 53(3):283 – 296, 2004.

[131] V. Popova and A. Sharpanskykh. Formal Analysis of Executions of Organizational Scenarios based on Process-oriented Specifications. *Applied Intelligence*, 34(2):226–244, April 2011.

[132] L.J. Porter and A.J. Parker. Total Quality Management – the Critical Success Factors. *Total Quality Management*, 4(1):13–22, 1993.

[133] F. Puhlmann and M. Weske. Interaction Soundness for Service Orchestrations. In *Proceedings of the 4th international conference on Service-Oriented Computing*, ICSOC'06, pages 302–313, Berlin, Heidelberg, 2006. Springer-Verlag.

[134] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, March 1986.

[135] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[136] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257 –286, feb 1989.

[137] E. Ramezani, D. Fahland, and W.M.P. van der Aalst. Where did I Misbehave? Diagnostic Information in Compliance Checking. In A. Barros, A. Gal, and E. Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 262–278. Springer Berlin Heidelberg, 2012.

[138] E. Ramezani, D. Fahland, B.F. van Dongen, and W.M.P. van der Aalst. Diagnostic Information in Temporal Compliance Checking. In C. Salinesi, M.C. Norrie, and O. Pastor, editors, *Advanced Information Systems Engineering*, volume 7908 of *Lecture Notes in Computer Science*, pages 304–320. Springer Berlin Heidelberg, 2013.

[139] A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. 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 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer Berlin / Heidelberg, 2003.

[140] A. Rebuge and D.R. Ferreira. Business Process Analysis in Healthcare Environments: A Methodology based on Process Mining. *Information Systems*, 37(2):99–116, April 2012.

[141] M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in Process-Aware Information Systems. In K. Jensen and W.M.P. van der Aalst, editors, *Transactions on Petri Nets and Other Models of Concurrency II*, pages 115–135. Springer-Verlag, Berlin, Heidelberg, 2009.

[142] H. A. Reijers, T. Slaats, and C. Stahl. Declarative Modeling-An Academic Dream or the Future for BPM? In F. Daniel, J. Wang, and B. Weber, editors, *Proceedings of 11th International Conference on Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2013.

[143] H.A. Reijers and I.T.P. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In J. Desel, B. Pernici, and M. Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 290–305. Springer Berlin Heidelberg, 2004.

[144] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes ofWeb Services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '06, pages 205–212, Washington, DC, USA, 2006. IEEE Computer Society.

[145] A. Rozinat. *Process Mining: Conformance and Extension*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2010.

[146] A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Conformance Analysis of ASML's Test Process. In *Proceedings of the Second International Workshop on Governance, Risk and Compliance (GRCIS'09)*, volume 459, pages 1–15. CEUR-WS.org, 2009.

[147] A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Process Mining Applied to the Test Process of Wafer Steppers in ASML. *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 39:474–479, 2009.

[148] A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The Need for a Process Mining Evaluation Framework in Research and Practice. In A. ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 84–89. Springer-Verlag, Berlin, 2008.

[149] A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering Simulation Models. *Information Systems*, 34(3):305 – 327, 2009.

[150] A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In S. Dustdar, J.L. Fiadeiro, and A.P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer Berlin Heidelberg, 2006.

[151] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33:64–95, March 2008.

[152] A. Rozinat, M. Veloso, and W.M.P. van der Aalst. Using Hidden Markov Models to Evaluate the Quality of Discovered Process Models. Technical report, BPMcenter.org, 2008. BPM Center Report BPM-08-10.

[153] A. Scheer. *Aris-Business Process Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2000.

[154] K. Schmidt. Stubborn Sets for Standard Properties. In *Proceedings of the 20th International Conference on Application and Theory of Petri Nets*, pages 46–65, London, UK, UK, 1999. Springer-Verlag.

[155] K. Schmidt. Stubborn Sets for Model Checking the EF/AG fragment of CTL. *Fundamenta Informaticae*, 43(1-4):331–341, August 2000.

[156] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst. Process Flexibility: A Survey of Contemporary Approaches. In J.L.G. Dietz, A. Albani, and J. Barjis, editors, *CIAO! / EOMAS*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer, 2008.

[157] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[158] M. Schroeck, R. Shockley, J. Smart, D. Romero-Morales, and P. Tufano. Analytics: The Real-World use of Big Data. `https://www14.software.ibm.com/webapp/iwm/web/signup.do?source=csuite-NA&S_PKG=Q412IBVBigData`, 2012. Last accessed 19th Jan 2013.

[159] S. Shetty. Gartner Executive Programs Survey of CIOs in India Identifies Cloud Computing as the Top Technology Priority for CIOs in 2011. `http://www.gartner.com/it/page.jsp?id=1574514`, 2011. last accessed on 17th January 2013.

[160] P. Soffer. On the Notion of Flexibility in Business Processes. In *Proceedings of CAiSE'05 Workshops*, pages 35–42, 2005.

[161] 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 17th Workshop on Information Technologies and Systems (WITS'07), Montreal, Canada*, pages 139–145, 2007.

[162] J. Swinnen, B. Depaire, M. Jans, and K. Vanhoof. A Process Deviation Analysis – A Case Study. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 87–98. Springer Berlin Heidelberg, 2012.

[163] D. Tam. Facebook Processes More Than 500 TB of Data Daily. `http://news.cnet.com/8301-1023_3-57498531-93/facebook-processes-more-than-500-tb-of-data-daily/`, August 2012. last accessed on 20th January 2013.

[164] J.J. Thomas and K.A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, Los Alametos, CA, 2005.

[165] J.S. Trefil. *Cassell's Laws of Nature*. Cassell Reference, 2002.

[166] A.L. Tucker and A.C. Edmondson. Managing Routine Exceptions: A Model of Nurse Problem Solving Behavior. *Advances in Health Care Management*, 3:87–113, 2002.

[167] W.M.P. van der Aalst. Three Good Reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, 1996.

[168] W.M.P. van der Aalst. Verification of Workflow Nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, London, UK, 1997. Springer-Verlag.

[169] W.M.P. van der Aalst. The Application of Petri nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[170] W.M.P. van der Aalst. Business Alignment: using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering*, 10:198–211, November 2005.

[171] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Verlag, 2011.

[172] W.M.P. van der Aalst. Decomposing Process Mining Problems Using Passages. In S. Haddad and L. Pomello, editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 72–91. Springer Berlin Heidelberg, 2012.

[173] W.M.P. van der Aalst. Decomposing Petri nets for Process Mining: A Generic Approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[174] W.M.P. van der Aalst. Mediating between Modeled and Observed Behavior: The Quest for the Right Process. In *Proceedings of the 7th IEEE International Conference on Research Challenges in Information Science (RCIS), 2013*, pages 1–12, 2013.

[175] W.M.P. van der Aalst, A. Adriansyah, A.K.A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, R.P.J.C Bose, P. van den Brand, R. Brandtjen, J.C.A.M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B.F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C.W. Günther, A. Guzzo, P. Harmon, A.H.M. ter Hofstede, J. Hoogland, J.E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F.M. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R.M. Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H.S. Pérez, R.S. Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K.D. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M.T. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer Berlin Heidelberg, 2012.

[176] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Causal Nets: a Modeling Language Tailored Towards Process Discovery. In *Proceedings of the 22nd International Conference on Concurrency Theory*, CONCUR'11, pages 28–42, Berlin, Heidelberg, 2011. Springer-Verlag.

[177] 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.

[178] 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, *Lecture Notes in Computer Science, 3760: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005*, pages 130–147. Springer-Verlag, October 2005.

[179] W.M.P. van der Aalst and A.K.A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. *Electronic Notes in Theoretical Computer Science*, 121:3 – 21, 2005. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).

[180] W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H.M.W. Verbeek. Choreography Conformance Checking: An Approach based on BPEL and Petri Nets (extended version). Technical report, BPMcenter.org, 2005.

[181] W.M.P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Núñez, and G. Zavattaro, editors, *WS-FM*, pages 1–23. Springer, 2006.

[182] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work*, 14(6):549–593, December 2005.

[183] W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time Prediction based on Process Mining. *Information Systems*, 36(2):450–475, April 2011.

[184] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[185] W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *Proceedings of the 2003 international conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[186] W.M.P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

[187] W.M.P. van der Aalst, K. M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, May 2011.

[188] 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, March 2010.

[189] R. van der Toorn. *Component-Based Software Design with Petri nets: An Approach Based on Inheritance of Behavior*. PhD thesis, Eindhoven University of Technology, 2004.

[190] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informatica.*, 94(3-4):387–412, 2009.

[191] J.M.E.M van der Werf, H.M.W. Verbeek, and W.M.P. van der Aalst. Context-Aware Compliance Checking. In A. Barros, A. Gal, and E. Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 98–113. Springer Berlin Heidelberg, 2012.

[192] B.F. van Dongen. Event Log for the BPI Challenge 2012, 2012.

[193] B.F. van Dongen and A. Adriansyah. Process Mining: Fuzzy Clustering and Performance Visualization. In S. Rinderle-Ma, S.W. Sadiq, and F. Leymann, editors, *Business Process Management Workshops, BPM 2009 International Workshops, Ulm, Germany, September 7, 2009. Revised Papers*, volume 43 of *Lecture Notes in Business Information Processing*, pages 158–169. Springer, 2009.

[194] S. vanden Broucke, J. de Weerdt, J. Vanthienen, and B. Baesens. A Comprehensive Benchmarking Framework (CoBeFra) for Conformance Analysis between Procedural Process Models and Event Logs in ProM. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, part of the IEEE Symposium Series on Computational Intelligence 2013, SSCI 2013*, volume accepted, 2013.

[195] S.K.L.M. vanden Broucke, J. de Weerdt, B. Baesens, and J. Vanthienen. Improved Artificial Negative Event Generation to Enhance Process Event Logs. In J. Ralytè, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 254–269. Springer Berlin Heidelberg, 2012.

[196] S.K.L.M. vanden Broucke, J. de Weerdt, J. Vanthienen, and B. Baesens. On Replaying Process Execution Traces Containing Positive and Negative Events. Technical report, KU Leuven - Faculty of Economics and Business, 2013. FEB Research Report KBI_1311.

[197] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*, volume 37 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishing, 2001.

[198] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The Process Mining Toolkit. In M. La Rosa, editor, *Proceedings of BPM Demonstration Track 2010*, volume 615 of *CEUR Workshop Proceedings*, pages 34–39, Hoboken, USA, September 2010. CEUR-WS.org.

[199] H.M.W. Verbeek and W.M.P. van der Aalst. Decomposing Replay Problems: A Case Study. BPM Center Report BPM-13-09, BPMcenter.org, 2013.

[200] H.M.W. Verbeek, M.T. Wynn, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Reduction Rules for Reset/Inhibitor Nets. *Journal of Computer and System Sciences*, 76(2):125–143, March 2010.

[201] B. Weber, H.A. Reijers, S. Zugal, and W. Wild. The Declarative Approach to Business Process Execution: An Empirical Test. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 470–485, Berlin, Heidelberg, 2009. Springer-Verlag.

[202] M. Weidlich, A. Polyvyanyy, N. Desai, and J. Mendling. Process Compliance Measurement based on Behavioural Profiles. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering*, CAiSE'10, pages 499–514, Berlin, Heidelberg, 2010. Springer-Verlag.

[203] M. Weidlich, H. Ziekow, J. Mendling, O. Gunther, M. Weske, and N. Desai. Event-Based Monitoring of Process Execution Violations. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 182–198. Springer Berlin Heidelberg, 2011.

[204] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the Heuristics Miner-algorithm. Technical report, Eindhoven University of Technology, Eindhoven, 2006. BETA Working Paper Series, WP 166.

[205] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, May 2012.

[206] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann. Monitoring and Analyzing Influential Factors of Business Process Performance. In *Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference*, EDOC '09, pages 141–150, Washington, DC, USA, 2009. IEEE Computer Society.

[207] B. Wetzstein, S. Strauch, and F. Leymann. Measuring Performance Metrics of WS-BPEL Service Compositions. In *Proceedings of the Fifth International Conference on Networking and Services*, pages 49 –56, april 2009.

[208] G. Winskel. Petri Nets, Algebras, Morphisms, and Compositionality. *Information and Computation*, 72(3):197 – 238, 1987.

[209] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives (revised version). BPM Center Report BPM-06-17, BPMcenter.org, 2006.

[210] M.D. Zisman. *Office Automation: Revolution or Evolution?* PhD thesis, Massachusets Institute of Technology, 1978.

[211] M. zur Muehlen and J. Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *Proceedings of the 20th International Conference on Advanced Information Systems Engineering*, CAiSE '08, pages 465–479, Berlin, Heidelberg, 2008. Springer-Verlag.

[212] M. zur Muehlen and M. Rosemann. Workflow-Based Process Monitoring and Controlling: Technical and Organizational Issues. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, volume 6 of *HICSS '00*, pages 6032–, Washington, DC, USA, 2000. IEEE Computer Society.

[213] M. zur Muehlen and R. Shapiro. Business Process Analytics. *Handbook on Business Process Management 2*, 2, 2009.

Summary

Aligning Observed and Modeled Behavior

The increased level of competition between organizations forces individual organizations to perform in the best way possible. Experience shows that business processes are one of the keys to improve the overall performance of organizations. However, improving business processes is not a trivial task. Business processes often comprise of many activities that involve many people across different organizations. Therefore, many organizations document their business processes as process models to provide insights into business processes in an effective and efficient way.

The rapidly changing business environment often forces people and organizations to be highly flexible and allow deviations from documented process models. Therefore, it is important to *align* the observed behavior occurring in reality to the behavior described in process models. Many organizations nowadays use information systems to support their business process. Such systems typically log all events that occurred during process executions, i.e., they record all *observed behavior*. This information can be exploited to identify deviations to documented process models and further extended to provide other types of analysis.

Many approaches to relate the observed behavior and the modeled behavior of a process have been proposed. However, many of these approaches either use heuristics that may yield misleading results or rely on assumptions that are difficult to satisfy. Consequently, the applicability of such approaches are rather limited. In this thesis, we investigate *robust approaches to align the observed behavior in an event log with the modeled behavior in a process model*. Furthermore, we investigate various extensions of the approaches that can be developed to obtain insights into process executions.

The contributions of this thesis can be divided into two main parts. First of all, the thesis contributes to the fundamentals of aligning observed and modeled behavior. Based on literature, we describe a set of requirements for approaches that align observed and modeled behavior. We formalize a notion of alignments that satisfies all of the requirements. Comparisons with existing approaches using the set of requirements show that the notion of alignments is more robust to peculiarities of process models and observed behavior in event logs. Given a trace of an event log and a process model, an alignment between them shows explicitly where deviations are and which activities in the trace cause the deviations.

Constructing alignments is a computationally expensive task. Hence, we also propose a memory-efficient approach to compute alignments based on the $\mathbb{A}^\star$ algorithm. Experimental results show that the approach is not only memory-efficient, but also time-efficient in cases where process models are complex and traces are both large and non-

fitting. Furthermore, we show that alignments can also be extended to explicitly show high-level deviations between a given trace and process model.

The notion of alignments is fundamental for various types of analysis. Next to providing the foundations for computing alignments, the thesis also shows how alignments can be applied:

- In the area of *conformance checking*, alignments are crucial to provide robust fitness and precision measurements between a given event log and a process model.
- In the area of *deviation analysis*, alignments show possible root causes of deviations between a log and a model. We provide three visualizations of alignments, each highlighting a different aspect of possible causes of deviations. Furthermore, we show using case studies that the combination of alignments and existing data mining techniques yields a powerful technique to analyze the possible root causes of deviations.
- In the area of *performance measurements*, we show that alignments are crucial for measuring performance based on event logs and process models. We show that alignment-based performance measurements provide more accurate performance measurements than existing approaches in literature in cases where the logs are non-fitting.

The work presented in this thesis has been evaluated using various case studies taken from real-life event logs and process models. Furthermore, the techniques presented in this thesis are implemented and publicly available as packages of the ProM framework (see `http://www.processmining.org`).

Samenvatting

Aligning Observed and Modeled Behavior

De toenemende competitie tussen organisaties dwingt individuele organisaties om op de best mogelijke manier te presteren. Ervaring leert dat bedrijfsprocessen een van de kernelementen zijn om de prestaties van organisaties te verbeteren. Het verbeteren van bedrijfsprocessen in echter geen eenvoudige taak. Bedrijfsprocessen omvatten meestal meerdere activiteiten welke meerdere mensen over verschillende organisaties betreft. Daarom documenteren veel organisaties hun bedrijfsprocessen in procesmodellen om op een effectieve en efficinte manier inzichten in hun bedrijfsprocessen te verschaffen.

De snel veranderende zakelijke omgeving dwingt mensen en organisaties vaak om flexibel te zijn en afwijkingen toe te staan van de gedocumenteerde procesmodellen. Daarom is het belangrijk om het geobserveerde echte gedrag *uit te lijnen* met het gedrag zoals beschreven in de procesmodellen. Veel organisaties gebruiken tegenwoordig informatiesystemen om hun bedrijfsprocessen te ondersteunen. Deze systemen registeren typisch alle gebeurtenissen die tijdens het uitvoeren van het proces voorkwamen, ze registreren dus al het *geobserveerd gedrag*. Deze informatie kan gebruikt worden om afwijkingen van de gedocumenteerde procesmodellen te identificeren en verder uitgebreid worden om andere typen analyses aan te bieden.

Veel verschillende aanpakken om geobserveerd gedrag aan het gemodelleerd gedrag te koppelen zijn reeds voorgesteld. Echter, de meeste van deze aanpakken gebruiken heuristieken die tot misleidende resultaten leiden, of ze doen aannamen die moeilijk zijn om aan te voldoen. Als een gevolg hiervan is de toepasbaarheid van deze aanpakken vrij beperkt. In dit proefschrift onderzoeken we *robuuste aanpakken om geobserveerd gedrag in een gebeurtenissenlogboek uit te lijnen met het gemodelleerde gedrag in het procesmodel*. Bovendien onderzoeken we verschillende uitbreidingen van deze aanpakken die ontwikkeld kunnen worden om inzicht te verkrijgen in de uitvoer van processen.

De contributie van dit proefschrift kan in twee hoofdonderdelen gesplitst worden. Allereerst draagt dit proefschrift bij aan de basis van het uitlijnen van geobserveerd en gemodelleerd gedrag. Op basis van literatuur beschrijven we een verzameling eisen voor aanpakken die geobserveerd en gemodelleerd gedrag uitlijnen. We formaliseren een concept van uitlijningen die aan elk van deze gestelde eisen voldoet. Vergelijkingen met bestaande aanpakken op basis van de set van eisen toont dat het concept van uitlijningen robuuster is voor eigenaardigheden van procesmodellen en geobserveerd gedrag in gebeurtenissenlogboeken. Gegeven een zaak in een gebeurtenissenlogboek en een procesmodel, een uitlijning tussen hen toont expliciet waar afwijkingen plaatsvinden en welke activiteiten in de zaak de oorzaak zijn van de afwijkingen.

Het maken van uitlijningen is een rekenkundig zware taak. Daarom stellen we ook een geheugen-efficinte aanpak om uitlijnen te berekenen voor op basis van het $\mathbb{A}^*$ algoritme. Experimentele resultaten tonen aan dat de aanpak niet alleen geheugen-efficint is maar ook tijd-efficint in gevallen met complexe procesmodellen en lange, slecht passende, zaken. Bovendien tonen we aan dat uitlijningen ook uitgebreid kunnen worden om op een hoger niveau afwijkingen tussen een gegeven zaak en het procesmodel aan te tonen.

Het concept van een uitlijning is fundamenteel voor verschillende typen analyse. Naast het bieden van de fundamenten voor het uitrekenen van uitlijningen toont dit proefschrift ook hoe uitlijningen toegepast kunnen worden:

- In het gebied van *conformiteit controle* zijn uitlijningen cruciaal om een robuuste fitheid en precisie tussen een gegeven gebeurtenissenlogboek en procesmodel aan te geven.
- In het gebied van *afwijking analyse* tonen uitlijningen mogelijke kernoorzaken voor afwijkingen tussen het logboek en een model. We presenteren drie visualisaties van uitlijningen waarbij elke een ander aspect van de mogelijke oorzaken van afwijkingen uitlicht. Verder tonen we door middel van praktijkonderzoeken dat de combinatie van uitlijningen en bestaande data-mining technieken een krachtige techniek oplevert om de kernoorzaak van afwijkingen te analyseren.
- In het gebied van *prestatie metingen* tonen we dat uitlijningen cruciaal zijn voor het meten van prestatie gebaseerd op gebeurtenissenlogboeken en procesmodellen. We tonen aan dat uitlijning-gebaseerde prestatie metingen meer accurate prestatie waarden geven dan bestaande aanpakken in literatuur in gevallen waarbij de logboeken niet-passend zijn.

Het werk gepresenteerd in dit proefschrift is gevalueerd op basis van verschillende praktijkonderzoeken gebruik makende van echte gebeurtenissenlogboeken en procesmodellen. Bovendien zijn de technieken zoals gepresenteerd in dit proefschrift gemplementeerd en publiek beschikbaar als modules in het ProM framework (zie `http://www.processmining.org`).

Acknowledgements

Alhamdulillahirabbil'aalamiin..

There are many people that I would like to thank. First of all, this dissertation does not exist if my promotor Wil van der Aalst did not offer me a PhD position in the AIS group. I am greatly thankful for all of his supervision throughout these 4 years. It has been an great pleasure for me to work with one of the most brilliant people in the world with such a great personality. I thank my co-promotor Boudewijn van Dongen for the wonderful time spent to discuss ideas, bits and bytes, "articles", and nasty data structures. I learned that on top of many hours of sweats and tears, a good research result also requires strong theoretical foundations, top programming skills, and smart marketing strategies. I also learned that a good researcher is driven by a passion to solve problems instead of a "passion" to have many papers on top-tier conferences or high h-index. I would further thank Emile Aarts, Bart Baessens, Josep Carmona, Uzay Kaymak, Milan Petkovic, and Alessandro Sperdutti for being the members of my graduation committee.

Next, I'd like to thank all people in the Information System/AIS group. In particular, I thank Natalia Sidorova for giving me another point of view of being a PhD student, Jan Martijn van der Werf for being such a nice office-mate/classical music DJ for almost 4 years, Joos Buijs for being such a great buddy (and thanks for helping me translating the summary of this thesis!), Eric Verbeek for solving ProM technical issues and stopping me printing papers (by giving me a nice huge screen), JC for integrating the trace alignment plug-in with the PNReplayer package, and of course both Ine van der Ligt and Riet van Buul for making all administration tasks look easy. Anne, Arjan, Carmen, Christian, Debjyoti, Delia, Dennis, Dirk, Elham, Fabrizio, Faisal, Felix, Hajo, Han, Helen, Jan, Joyce, Kees, Maja, Massimiliano, Michael, Michiel, Rafal, Richard, Ronny, Sander, Shengnan, and Xixi: thank you for making my PhD period memorable!

I am also fortunate enough to collaborate with wonderful people from outside of the AIS group. Among all people that I've worked with, I'd like to thank Jorge Munoz-Gama and Josep Carmona for the series of successful collaborations. Indeed, IT can really remove physical boundaries. I thank Nicola Zannone for the collaborations that resulted in a couple of (journal) papers. It is remarkable what can one achieve by combining ideas from different domains. I also thank Joel Ribeiro for the cooperation in the implementation of Flexible Heuristic Miner.

In March 2013, I had an opportunity to shortly visit the BPM group in QUT. I'd like to thank Marcello La Rosa for his supervision and Arthur ter Hofstede for his warm hospitality while I was staying in Brisbane. I thank the cheerful Raffaele Conforti for the hours of fun research collaborations. Many thanks to Artem, Jochen, Alex, Niz, and Julia for making me feel at home.

Apparently, academic life can also be stressful. Therefore, I thank everybody in Musihoven, PPI Eindhoven, and Angklung Eindhoven communities for all of the fun, laughter, and cheers that help me go through the blue days.

I realize that the list of people mentioned here is far from complete. Nevertheless, I'd like to thank you for everyone else that supports me until the end of my PhD.

I save the last paragraphs for people that are simply irreplaceable in my life. I'd like to thank my parents, Darmawan Daud and Meydia Hasan, who motivated me to do a PhD in the first place. Taking a PhD is not the easiest decision I've ever made, but I never regret that I took my chance. Your endless prayers and motivations are definitely one of the things that help me finished this thesis. I thank my sister, Astrid Hapsari Ningrum, for all of the silly things she did to make my life more colorful. I thank my parents in law, Furqon and Nunung Sobarningsih, for their prayers and supports. I also thank my brothers and sisters in law: Imron Rosyadi, Aisyah Mayuliani, and Radian Furqon for all of their supports from distance.

Last, but definitely one of the greatest gratitude has to go to my wonderful small family. My dearest wife Elva Fitriani, thank you for always supporting me and helping me think straight when things got tough. I am truly grateful to have you by my side. My dear daughter Fakhira Tifania Azzahra, thank you for all the miracles you bring to the family. This thesis is dedicated to both of you.

Arya Adriansyah
Eindhoven, February 2014

Curriculum Vitae

Arya Adriansyah was born on December 14, 1984 in Bandung, Indonesia. He attended high school at SMU Negeri 3 Bandung, Indonesia. After finishing high school as one of the best graduates in 2002, he started his studies at Bandung Institute of Technology, Indonesia. In this period, he was actively involved in many student organizations, participated in many IT competitions, and worked in various IT projects as a programmer/system analyst. He also had his name on the Dean's list for being one of the best students in the faculty of Industrial Engineering. He obtained a bachelor degree (Sarjana Teknik) in Informatics Engineering in 2006 (cum laude). Not long after he graduated, he started an academic career at the same institute as a research assistant. Among other courses, he was mainly involved in programming and software development courses as a lecturer.

In 2007, he received a full master scholarship in Computer Science from Eindhoven University of Technology (TU/e), the Netherlands. He did an internship in the Architecture of Information System Group and implemented many behavioral and structural process model analysis techniques. He did his thesis in the same group with the title "Performance Analysis of Business Processes from Event Logs and Given Process Models" under the supervision of Prof. dr. Wil van der Aalst and successfully defended the thesis in August 2009. He received a Master of Science (M.Sc.) degree in Computer Science and Engineering with distinction (cum laude) from TU/e in the end of the same month.

Just before his master graduation, he accepted a 4 year contract offer to work as a PhD candidate on the REPLAY project in the same university. This project is funded by NWO (a Dutch research institution). In November 2007, he started his PhD project to investigate approaches to align the observed behavior in an event log with the modeled behavior in a process model. His research results led to a number of publications in well-known international journals, conferences, and workshops. He implemented his findings, i.e., various alignment techniques, as publicly available plugins in the ProM framework. This way, the techniques can be used by people from either research communities or industries. He finishes his project in the end of 2013, which resulted in this thesis.

Since November 2013, he has been working in Accenture Netherlands as a junior consultant in the Emerging Technology and Innovation Group to follow his passion in becoming an IT expert. He can be reached in `arya.adriansyah@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 Perspective
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 Coup (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 politile 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 Ramrez Camps (CWI) Structural Features in XML Retrieval
2007-25 Joost Schalken (VU) Empirical Investigations in Software Process Improvement

2008

2008-01 Katalin Boer-Sorbn (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-Universitaet 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-Universitaet 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 Withdrawn
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 Trajectories
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
2012-13 Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
2012-14 Evgeny Knutov(TUE) Generic Adaptation Framework for Unifying Adaptive Web-based Systems
2012-15 Natalie van der Wal (VU) Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
2012-16 Fiemke Both (VU) Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
2012-17 Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance
2012-18 Eltjo Poort (VU) Improving Solution Architecting Practices
2012-19 Helen Schonenberg (TUE) What's Next? Operational Support for Business Process Execution
2012-20 Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
2012-21 Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval
2012-22 Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
2012-23 Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
2012-24 Laurens van der Werff (UT) Evaluation of Noisy Transcripts for Spoken Document Retrieval
2012-25 Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
2012-26 Emile de Maat (UVA) Making Sense of Legal Text
2012-27 Hayrettin Gurkok (UT) Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
2012-28 Nancy Pascall (UvT) Engendering Technology Empowering Women
2012-29 Almer Tigelaar (UT) Peer-to-Peer Information Retrieval
2012-30 Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making
2012-31 Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
2012-32 Wietske Visser (TUD) Qualitative multi-criteria preference representation and reasoning
2012-33 Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)
2012-34 Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications
2012-35 Evert Haasdijk (VU) Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
2012-36 Denis Ssebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes
2012-37 Agnes Nakakawa (RUN) A Collaboration Process for Enterprise Architecture Creation
2012-38 Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms
2012-39 Hassan Fatemi (UT) Risk-aware design of value and coordination networks
2012-40 Agus Gunawan (UvT) Information Access for SMEs in Indonesia
2012-41 Sebastian Kelle (OU) Game Design Patterns for Learning
2012-42 Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning
2012-43 Withdrawn
2012-44 Anna Tordai (VU) On Combining Alignment Techniques
2012-45 Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions
2012-46 Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
2012-47 Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior
2012-48 Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM) Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
2012-50 Steven van Kervel (TUD) Ontologogy driven Enterprise Information Systems Engineering
2012-51 Jeroen de Jong (TUD) Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013

2013-01 Viorel Milea (EUR) News Analytics for Financial Decision Support
2013-02 Erietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
2013-03 Szymon Klarman (VU) Reasoning with Contexts in Description Logics
2013-04 Chetan Yadati(TUD) Coordinating autonomous planning and scheduling
2013-05 Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns
2013-06 Romulo Goncalves(CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
2013-07 Giel van Lankveld (UvT) Quantifying Individual Player Differences
2013-08 Robbert-Jan Merk(VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
2013-09 Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications
2013-10 Jeewanie Jayasinghe Arachchige(UvT) A Unified Modeling Framework for Service Design.
2013-11 Evangelos Pournaras(TUD) Multi-level Reconfigurable Self-organization in Overlay Services
2013-12 Marian Razavian(VU) Knowledge-driven Migration to Services
2013-13 Mohammad Safiri(UT) Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
2013-14 Jafar Tanha (UVA) Ensemble Approaches to Semi-Supervised Learning Learning
2013-15 Daniel Hennes (UM) Multiagent Learning - Dynamic Games and Applications
2013-16 Eric Kok (UU) Exploring the practical benefits of argumentation in multi-agent deliberation
2013-17 Koen Kok (VU) The PowerMatcher: Smart Coordination for the Smart Electricity Grid
2013-18 Jeroen Janssens (UvT) Outlier Selection and One-Class Classification
2013-19 Renze Steenhuizen (TUD) Coordinated Multi-Agent Planning and Scheduling
2013-20 Katja Hofmann (UvA) Fast and Reliable Online Learning to Rank for Information Retrieval
2013-21 Sander Wubben (UvT) Text-to-text generation by monolingual machine translation
2013-22 Tom Claassen (RUN) Causal Discovery and Logic
2013-23 Patricio de Alencar Silva(UvT) Value Activity Monitoring
2013-24 Haitham Bou Ammar (UM) Automated Transfer in Reinforcement Learning
2013-25 Agnieszka Anna Latoszek-Berendsen (UM) Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
2013-26 Alireza Zarghami (UT) Architectural Support for Dynamic Homecare Service Provisioning
2013-27 Mohammad Huq (UT) Inference-based Framework Managing Data Provenance
2013-28 Frans van der Sluis (UT) When Complexity becomes Interesting: An Inquiry into the Information eXperience
2013-29 Iwan de Kok (UT) Listening Heads
2013-30 Joyce Nakatumba (TUE) Resource-Aware Business Process Management: Analysis and Support
2013-31 Dinh Khoa Nguyen (UvT) Blueprint Model and Language for Engineering Cloud Applications
2013-32 Kamakshi Rajagopal (OUN) Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
2013-33 Qi Gao (TUD) User Modeling and Personalization in the Microblogging Sphere
2013-34 Kien Tjin-Kam-Jet (UT) Distributed Deep Web Search
2013-35 Abdallah El Ali (UvA) Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA)
2013-36 Than Lam Hoang (TUe) Pattern Mining in Data Streams
2013-37 Dirk Brner (OUN) Ambient Learning Displays
2013-38 Eelco den Heijer (VU) Autonomous Evolutionary Art
2013-39 Joop de Jong (TUD) A Method for Enterprise Ontology based Design of Enterprise Information Systems
2013-40 Pim Nijssen (UM) Monte-Carlo Tree Search for Multi-Player Games
2013-41 Jochem Liem (UVA) Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
2013-42 Lon Planken (TUD) Algorithms for Simple Temporal Reasoning
2013-43 Marc Bron (UVA) Exploration and Contextualization through Interaction and Concepts

2014

2014-01 Nicola Barile (UU) Studies in Learning Monotone Models from Data
2014-02 Fiona Tuliyano (RUN) Combining System Dynamics with a Domain Modeling Method
2014-03 Sergio Raul Duarte Torres (UT) Information Retrieval for Children: Search Behavior and Solutions
2014-04 Hanna Jochmann-Mannak (UT) Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
2014-05 Jurriaan van Reijsen (UU) Knowledge Perspectives on Advancing Dynamic Capability
2014-06 Damian Tamburri (VU) Supporting Networked Software Development
2014-07 Arya Adriansyah (TUE) Aligning Observed and Modeled Behavior
2014-08 Samur Araujo (TUD) Data Integration over Distributed and Heterogeneous Data Endpoints
2014-09 Philip Jackson (UvT) Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language