Comparative process mining

Comparative Process Mining: Analyzing Variability in Process Data

Alfredo José Bolt Iriondo

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Bolt Iriondo, Alfredo José

Comparative Process Mining: Analyzing Variability in Process Data by A.J. Bolt Iriondo.
Eindhoven: Technische Universiteit Eindhoven, 2023. Proefschrift.

Cover design by Camilo Ordenes

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

ISBN 978-90-386-5641-0

Keywords: Process Cube, Process Mining Workflow, Process Comparison, Process Variability

SIKS Dissertation Series No. 2023-01. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by Ipskamp Printing B.V., Auke Vleerstraat 145, 7547 PH // Enschede, The Netherlands.

Comparative Process Mining:
Analyzing Variability in Process Data

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 11 januari 2023, om 11.00 uur

door

Alfredo José Bolt Iriondo

geboren te Santiago, Chili.

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof. dr. E.R. van den Heuvel	
1e promotor:	prof. prof. h. c. dr. h. c. dr. ir. W. M. P. van der Aalst	(RWTH Aachen)
2e promotor:	prof. dr. ir. H. A. Reijers	
leden:	prof. dr. I. Weber	(TU Berlin)
	prof. dr. N. Meratnia	
	prof. dr. M. E. Sepúlveda	(Pontificia Universidad Católica de Chile)
adviseur(s):	prof. dr. R. M. Dijkman	

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

“Sometimes science is more art than science... A lot of people don’t get that”

Rick Sánchez

Abstract

Modern organizations may have a large number processes with different characteristics. Most of them are supported by information systems ranging from excel sheets to ERP systems. Such systems leave a data footprint that consists of recorded executions of processes i.e., *event data*.

Process mining is a relatively young research discipline that is concerned with discovering, monitoring and improving real processes by extracting knowledge from event data readily available in today's systems [156]. Process mining supports the extraction of insights from data about the overall and inner behavior contained in any given process. Hundreds of different process mining techniques have been proposed in literature. These are not limited to process-model discovery and the checking of conformance. Also, other perspectives (e.g., data) and operational support (e.g., predictions) are included.

In real-life, business processes are not static: They have to adapt to constant environment changes (e.g., customer preferences, legal regulations, new competitors). Like any live species, organizations (and their business processes) also evolve according to Darwinian evolution: The best to adapt is the one that thrives. It is not uncommon for organizations that the same business process has to adapt to different contexts simultaneously, which leads to variability in such processes, and ultimately to different *process variants*.

In many scenarios, splitting a process into *variants* can effectively reduce its variability (hence, its complexity), making them easier to analyze. It also enables many types of analysis e.g., comparing the different variants of the process in order to identify the best practices and detect differences and similarities between variants. Nevertheless, the best way to split a process is not always clear.

In general, we observe execution data of a process without knowing much about it (sometimes we do not even know if there is an actual process in there). The best way to split and analyze unknown data is obscured, and it requires extensive trial-and-error experimenting until an acceptable solution is found.

This thesis addresses the problem of analyzing process variability by proposing techniques and tools that use event data to identify *variants* within a process, split them, compare them, and automate their analysis. Concretely, this thesis proposes the following contributions to the body of scientific knowledge:

A technique to support the interactive and consistent exploration of process variants (Chapter 3). Process Cubes are the result of adapting OLAP-operations to explore process data, where each cell in a process cube contains events that can be converted into a process variant. Process cubes enable the consistent exploration of process variants, which can be used by other process mining techniques.

A technique to compare process variants. (Chapter 5). This technique is able to compare process variants in terms of behavior (using event dimensions) and in terms of business rules, based on their event logs. The results are projected into a process model that serves as a "map" in which the differences are clearly identified and can be pinpointed to specific parts (e.g., activities) of the process.

A technique to detect relevant process variants in a general setting. (Chapter 6). This technique is able to detect process variants in process data by splitting cases based on the data attributes of their events. It uses statistical testing and unbiased variable selection to detect only relevant process variants. The result is a summary of relevant splittings, where each splitting leads to a set of variants. Each variant is then encoded into its corresponding set of traces. As a result, an enriched event log can be used by a process cube to split the event data into such variants using variant-related dimensions.

Support the execution of process mining workflows. (Chapter 4). The concept of a process mining workflow as a chain of process mining (and/or non-process mining) analysis steps is introduced. Process mining workflows can be used to completely describe arbitrary process mining experiments. Therefore, it enables full reproducibility of the results. The tool supporting these workflows is introduced and it is applied in several use cases.

Develop replicable and sound benchmarks. (Chapters 7 and 8). Process mining workflows are used to define two frameworks: one for benchmarking process discovery techniques and the other for benchmarking concept drift detec-

tion techniques. These frameworks are not meant only for comparing techniques: they also allow for benchmarking techniques at a statistical level for specific process characteristics, or to perform parameter sensitivity analysis. For example, the effect of parallelism on the quality of models produced by a process discovery technique can be studied.

For each of these contributions, a prototype software tool has been implemented. They are all publicly available to use.

Contents

List of Figures

List of Tables

Part I

Opening

Chapter 1

Introduction

The notion of a *process* is not recent. One of the earliest and most famous references to the concept of a process was elaborated by the Scottish economist Adam Smith in the late eighteen century through the following example:

> "One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head: to make the head requires two or three distinct operations: to put it on is a particular business, to whiten the pins is another... and the important business of making a pin is, in this manner, divided into about eighteen distinct operations, which in some manufactories are all performed by distinct hands, though in others the same man will sometime perform two or three of them."

Even though this example was aimed to illustrate the division of labor, it was one of the first to hint about the existence of individual *tasks* that could be performed by different people, and that a combination of those tasks can produce an *output* (e.g., a pin).

Modern definitions of a process now include several other components, such as *inputs*, *resources*, and *customers*. In the early 90's, Thomas Davenport [41] defined a process as:

> "A structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on *how* work is done within an organization, in contrast to a product focus that has an emphasis on *what* work is done. A

> process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action."

Although many other definitions with different flavors have been proposed in literature, the main ideas of Davenport are still relevant.

Within most organizations, many processes interact with each other. In the late 80's, Michael Porter was one of the first to 'organize' an organization in terms of its processes. In his renowned *Value Chain* model [123] Porter divides an organization into two types of processes: primary and support. Primary processes relate to the business core (e.g., inbound and outbound logistics, operations, marketing, sales, and services) and they define the way in which value is added to the products or services provided by the organization. Support processes aim to support the business core (e.g., human resource management, technology management, procurement, and infrastructure management). According to Porter, the competitive advantages of an organization over its competitors are located within the primary processes, while the support processes ensure that these advantages are sustainable. Today, it is widely-accepted that managing processes is the key for the success of an organization.

Modern organizations may have a large number processes with different characteristics. Most of them are supported by information systems ranging from excel sheets to ERP systems. Such systems leave a data footprint that consists of recorded executions of processes i.e., *event data*.

Process mining is a relatively young research discipline that is concerned with discovering, monitoring and improving real processes by extracting knowledge from event data readily available in today's systems [156]. Process mining supports the extraction of insights from data about the overall and inner behavior contained in any given process. Hundreds of different process mining techniques have been proposed in literature. These are not limited to process-model discovery and the checking of conformance. Also, other perspectives (e.g., data) and operational support (e.g., predictions) are included.

In real-life, business processes are not static: They have to adapt to constant environment changes (e.g., customer preferences, legal regulations, new competitors). Like any live species, organizations (and their business processes) also evolve according to Darwinian evolution: The best to adapt is the one that thrives. It is not uncommon for organizations that the same business process has to adapt to different contexts simultaneously, which leads to variability in such processes. Moreover, processes can change over time. This is known as *concept drift*.

Take for example a building permit application process in five different municipalities in the Netherlands [23–27]. In theory, the process is the same: a building permit is requested by an applicant, and the municipality has to analyze the request and decide whether to approve it or not. However, data shows us that the municipalities are doing things differently. Table 1.1 shows the number of different event classes (i.e., activities) performed in each municipality.

We can observe that the municipalities have a similar number of event classes (i.e., ranging from 331 to 381). However, if we combine the event data of the five municipalities, we can find a higher number of different event classes (i.e., 461). This can mean that each municipality has a set of activities that are not executed by other municipalities.

Let's consider that the way each municipality executes the process is a *variant* of the process itself, so that if there are five municipalities, there are also five variants of the process. If we combine these five variants into one mega-process, then we can obtain one single dataset that describes the combined behavior of the five variants of the process. Note that the complexity of this combined process (i.e., the number of different activities, and the possible relations between them) is much higher than the complexity of the individual variants. It is important to note that the complexity of the combined process is caused by two factors: The variability between the process variants that were merged, and by the complexity of the variants themselves. Therefore, we can reduce process complexity by reducing process variability.

In many scenarios, splitting a process into *variants* can effectively reduce its variability (hence, its complexity), making them easier to analyze. It also enables many types of analysis e.g., comparing the different variants of the process

Table 1.1: Distinct event classes (i.e., activities) observed in the execution data of a building permit application process in five different Dutch municipalities.

Municipality	# Event Classes
Muni. 1 [23]	381
Muni. 2 [24]	376
Muni. 3 [25]	369
Muni. 4 [26]	331
Muni. 5 [27]	352
Muni. 1 to 5 (combined)	**461**

in order to identify the best practices and detect differences and similarities between variants. Nevertheless, the best way to split a process is not always clear. In the case of the building permit application process, we know from domain knowledge that there are five municipalities, hence it makes sense to split the process in such way. However, such domain knowledge is not always available. In general, we observe execution data of a process without knowing much about it (sometimes we do not even know if there is an actual process in there). The best way to split and analyze unknown data is obscured, and it requires extensive trial-and-error experimenting until an acceptable solution is found.

This thesis addresses the problem of analyzing process variability by proposing techniques and tools that use event data to identify *variants* within a process, split them, compare them, and automate their analysis.

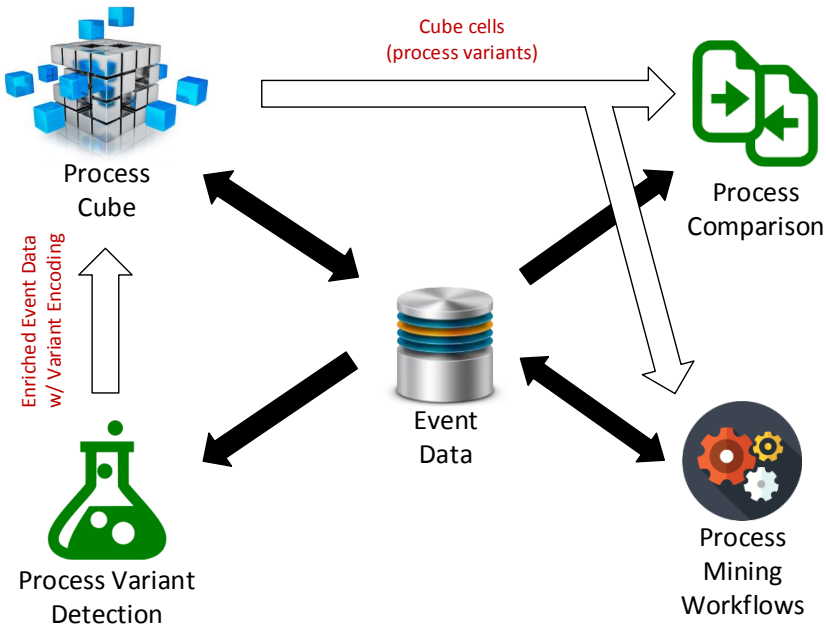


Figure 1.1: Overview of the concepts included in this thesis and the interactions between them.

Figure 1.1 illustrates the overall idea. The cornerstone of the techniques and tools presented in this thesis are *event data* (i.e., the observed execution data of processes) which can be stored in e.g., databases, documents, smart devices, spreadsheets, and event logs.

The event data are described by several data dimensions that characterize each execution. Dimensions can be *predefined* (i.e., given in the data) or can be *derived* from the context of the process and/or other predefined dimensions. Common examples of predefined dimensions are the *time* in which an execution was observed, the *resource* that performed the execution, and the specific *activity* that was executed. Examples of dimensions that are derived from the context of a process are the workload of resources, the type of customer, etc.

Such data dimensions can be used to split the event data into process variants by using process cubes, as illustrated in Figure 1.2. A cell in a process cube is defined by a combination of dimension values. Events are distributed into the cells of a process cube according to their values for such dimensions. For example, in Figure 1.2, a cube is defined by three dimensions (i.e., dimensions 1 to 3). The cell that is highlighted in red contains all the events that have a value "B" for *dimension 1*, a value β for *dimension 2*, and a value 1 for *dimension 3*.

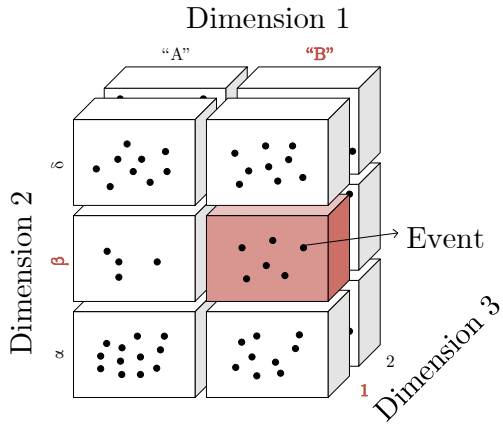


Figure 1.2: Abstract example of a process cube: each cell of the cube is defined by a specific combination of dimension values. Events are distributed into the cells according to their dimension values.

Predefined data dimensions do not always have a clear correlation with variability in the data. For example, if we randomly split event data using a dimension, it is possible that no significant variability is observed between the resulting process variants. If only predefined data dimensions are used to split the data into process variants, many opportunities for uncovering hidden variability in the process can be missed. In such cases, new data dimensions that have a strong correlation to process variability can be *derived* e.g., using process variant detection techniques. Then, event data can be enriched with these derived dimensions These derived dimensions can be used to split the event data (e.g., using a process cube) in such a way that the maximum amount of variability is exposed when comparing them, e.g., using process comparison techniques.

Event data and its process variants can also be used by process mining workflows, which can be used to automate their pre-processing and analysis. In this way, experiments can be performed in less time since user interaction is minimized. Also, experiments become more easily repeatable by other analysts and researchers. This thesis proposes scientific contributions in each one of these points.

It is important to note that the techniques proposed in this thesis (and illustrated in Figure 1.1) can also be used iteratively and independently, and do not necessarily have to be used in a specific order.

The remainder of this chapter is organized as follows. Section 1.1 provides an introductory overview of the field of process mining. Section 1.2 discusses the problem of variability in processes and how this thesis addresses it. Section 1.3 discusses the opportunities for automating process analysis. Section 1.4 describes the scientific contributions included in the thesis. Section 1.5 describes the structure of the thesis.

1.1 Process Mining

Process-mining techniques enable the analysis of a wide variety of processes using event data. The open-source process mining framework *ProM* [172] provides hundreds of plug-ins and has been downloaded over 150.000 times.[1] Nowadays, there are over 30 process mining software vendors (e.g., Disco, Perceptive Process Mining, Celonis Process Mining, QPR ProcessAnalyzer, Software AG/ARIS PPM, Fujitsu Interstage Automated Process Discovery, Minit, MyInvenio, etc.) working with small-to-large-sized companies in several countries

[1] ProM tools is free to download from `http://www.promtools.org`

worldwide. The formation of the *IEEE Task Force on Process Mining* is a reflection of the growing impact of process mining in the world. See for example the twenty *case studies* on the webpage of the IEEE Task Force on Process Mining [72].

Figure 1.3 illustrates the three different flavors in process mining: process discovery, conformance checking and process enhancement [156]. Process *dis-*

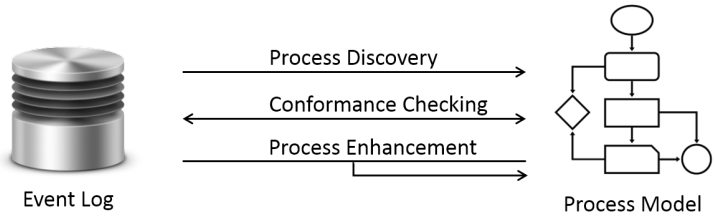


Figure 1.3: Process Mining in a nutshell.

covery techniques aim to obtain process models from event logs. Process models (hand-made or discovered) can be checked for *conformance* with respect to event logs. In this way, behavioral differences between the model and the real data can be detected. Process models can also be *enhanced* (i.e., improved or extended) using information about the actual process recorded in event logs.

In order to explain the different types of process mining techniques, we introduce an example fragment of an event log that relates to a journal reviewing process, shown in Figure 1.4. The process starts when an article submission is received. Then, the editor invites three reviewers. Each reviewer makes a revision of the article and sends it to the editor. However, there is a time limit

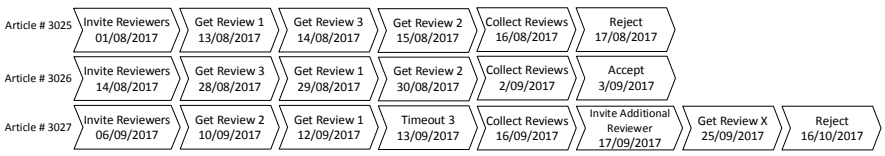


Figure 1.4: Example of a fragment of an event log containing a sample of executions of a journal reviewing process. Events are recorded for each executed activity and are grouped with other events related to the same execution of the process (called *case*) i.e., the same article.

for this. If the reviewer does not send the revision before such limit, a time-out occurs. After the reviews have been sent (or timeouts have occurred) the editor collects the reviews and decides whether an additional reviewer is needed or not. This step can happen more than once. Finally, the editor accepts or rejects the article.

The remainder of this section discusses and illustrates the three types of process mining techniques described above using this example process.

Process Discovery

The recorded executions of a process contained in an event log can provide information that is not only useful to analyze such executions individually; it can also be aggregated to obtain a global understanding of the whole process.

Process discovery techniques can produce a *model* of a process by only using an event log as input.

Definition 1.1 (Process Model)**.** *A process model is an abstract representation of a process from a defined* perspective *using a combination of elements of a defined* notation *(i.e., language)*.

When a building is designed, many blueprints i.e., models (e.g., structural, electrical, water, gas) are made from different perspectives: from a top view, from a side view, etc. They are necessary because a single blueprint does not contain all the information that is needed to construct a building. For example, a top view does not show the height of the building.

In processes, a similar phenomenon occurs. Different perspectives capture different information about the process, and no single perspective can capture everything about a process. Some of the most common process perspectives discussed in literature are:

- Control-flow: focus on the ordering and dependencies between activities.
- Performance: focus on time and how fast or slow is the execution of the process.
- Resource: focus on the persons, systems or machines that perform the activities.
- Data: focus on the data properties of events.

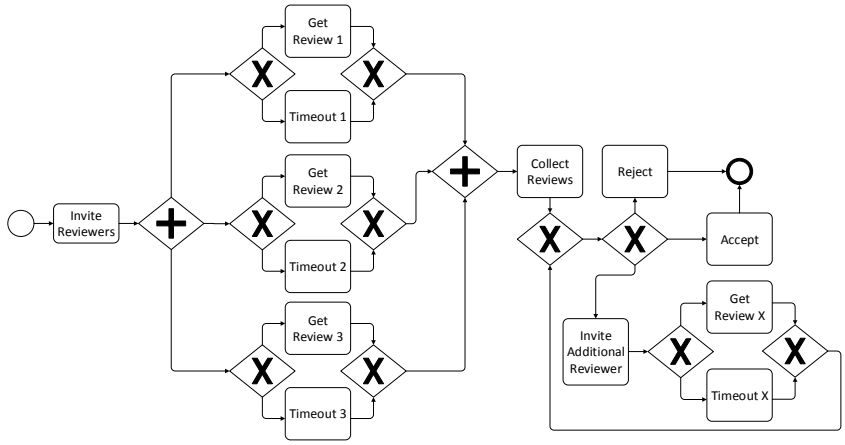


Figure 1.5: Process model in BPMN notation that represents the *control-flow* of the executions of the journal reviewing process.

A process modeling notation is simply a language of graphical elements that can be combined to represent a process. Naturally, process perspectives relate to specific modeling notations that contain elements that are specific to that perspective. For example, the control-flow perspective of a process can be modeled in many notations such as transition systems, Petri nets, BPMN, and many more, which contain elements to represent activities, choices, etc.

Figure 1.5 shows an example of a *control-flow* process model in *BPMN* notation that represents the recorded executions of the journal reviewing process illustrated in Figure 1.4. An extended discussion on control-flow notations is presented in the next chapter (see Section 2.3).

Conformance Checking

Process models can be obtained from event logs through process discovery techniques, or can be hand-made by analysts to state the "ideal" way the process should be. Process models might not always represent the way that a process is actually executed. For example, a process discovery technique might omit infrequent behavior, or an analyst could miss some special cases when designing the process model.

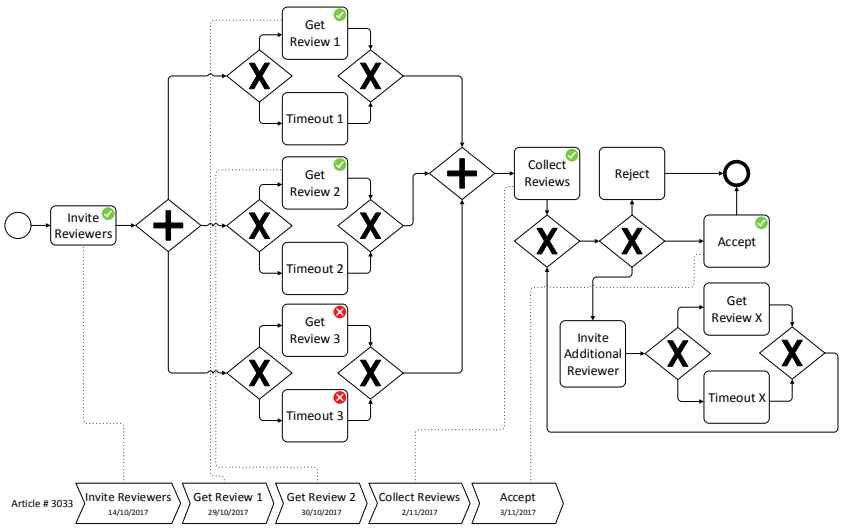


Figure 1.6: Example of conformance checking between the process model shown in Figure 1.5 and a new execution of the simplified journal reviewing process.

Conformance checking techniques can identify and quantify discrepancies between a process model and the real execution of the process i.e., the event log. Figure 1.6 illustrates the notion of conformance checking by an example. In this case, a new execution of the journal reviewing process is observed, consisting of only five activities: `invite reviewers`, `get review 1`, `get review 2`, `collect reviews`, and `accept`. If this is compared to the process model shown in Figure 1.5, we can observe a discrepancy: the model states that before the reviews can be collected, all three reviews have to be received or timed out). However, this is not observed in Figure 1.6. Given a discrepancy, analysts have to decide whether the model should incorporate such discrepant behavior or not. In this case, one of the two activities: `get review 3` and `timeout 3`, clearly should have been executed, as it is wrong to accept an article without collecting all the relevant reviews. In some other cases, the execution of the process might be indeed correct (e.g., special cases) and the model should be modified to incorporate such behavior.

Process Enhancement

Discovered process models rarely can provide sufficient insights to fully understand the process: most discovery techniques only capture the control-flow perspective. Process models can also be enhanced using the information contained in event logs. Figure 1.7 shows an example of an enhanced process model. Note that now activities contain information about their average frequency of occurrence in the event log (highlighted in blue), and also the average elapsed time since the case started until such activity is executed within the case (highlighted in red). These enhanced models provide a more complete view of the process, and can be used for several types of analysis.

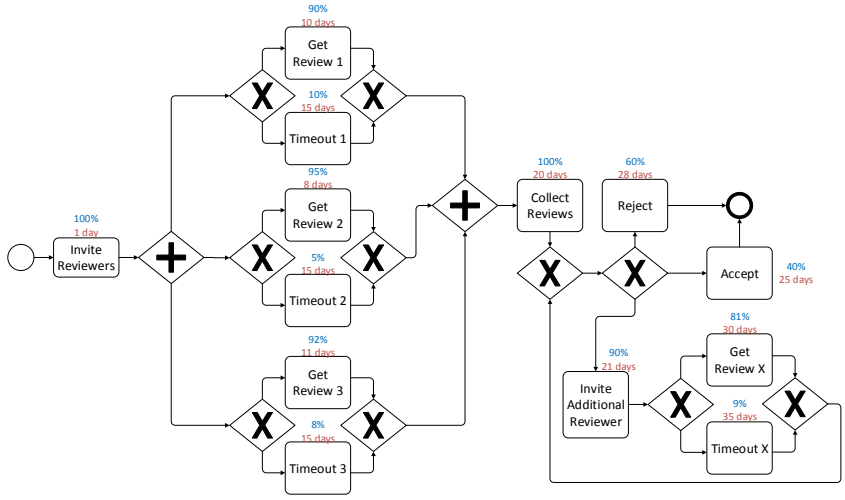


Figure 1.7: Example of a process model (shown in Figure 1.5) that has been *enhanced* using an event log . Blue numbers represents the percentage of cases that execute an activity. Red numbers represent the average elapsed time of cases for a given activity.

Now that we have introduced the basic concepts of process mining, we will further discuss the problem of variability presented before in more depth, as follows.

1.2 Dealing with Variability in Processes

In literature, variability in processes is usually related to the *control-flow* perspective (e.g., a process may skip risk assessment steps for gold customers), but can also be related to other perspectives, such as *performance*, *resources* and *data*. For example, if two branches of a company execute their processes in the same way (i.e., same control-flow) but there are huge performance differences between the branches, it is interesting to understand and explain such differences.

Naturally, too much variability can complicate the analysis of a process. The importance of dealing with variability is reflected in the process mining manifesto [152] as the challenges "*Dealing with complex event logs having diverse characteristics*" and "*Dealing with concept drift*". Note that concept drift refers to variability over time, e.g., the typical ordering of activities may change and bottlenecks may shift to different parts of the process.

The different process perspectives are usually materialized in event data in the form of data dimensions (i.e., attributes). The core concept in which this thesis is based, is that *the variability in processes can be reduced by splitting the data into variants using such dimensions*.

Event data can be split using a process cube (as shown in Figure 1.2). The cells of a process cube are defined by a specific combination of dimension values. Events are related to cells according to their values for such dimensions, in such a way that events with the same dimension values are grouped together into process variants.

However, as mentioned before, predefined dimensions included in the event data may not always unveil all the variability present in the process. Sometimes, new dimensions can be *derived* from other predefined (i.e., *given*) dimensions and context information.

Such dimensions can be used to split and explore the data differently in order to expose variability. For example, we could derive a "customer type" dimension from existing dimensions related to purchase history, risk factor, etc. Such derived dimension might have a stronger correlation to variability (e.g., VIP customers lead to shorter processing times) than the given dimensions it was derived from.

The remainder of this section discusses how variability is currently handled in process mining in each of the process perspectives defined previously (i.e., control-flow, performance, resource, and data), and sketches how the contributions introduced in this thesis may help dealing with variability.

Control-flow Variability

In [156], van der Aalst proposes to classify processes depending on their degree of structure. Processes can range from highly-structured *lasagna processes* to highly-unstructured *spaghetti processes*.

However, this process classification was originally conceived based only on the control-flow perspective. On the one hand, if the control-flow process model (designed or discovered) is relatively structured, then the process is considered as "lasagna". Note that "lasagna" processes tend to be very simple and structured, and most process mining techniques work well with this type of processes.

On the other hand, if the control-flow process model is highly complex, illegible, with many activities and lines connecting them, then the process is considered as "spaghetti". Hospitals are usually good examples of unstructured processes because of the high variety of exams and treatments applied to patients. Figure 1.8 shows a spaghetti process model that describes the process of an oncologic gynecology department at a Dutch hospital [171].

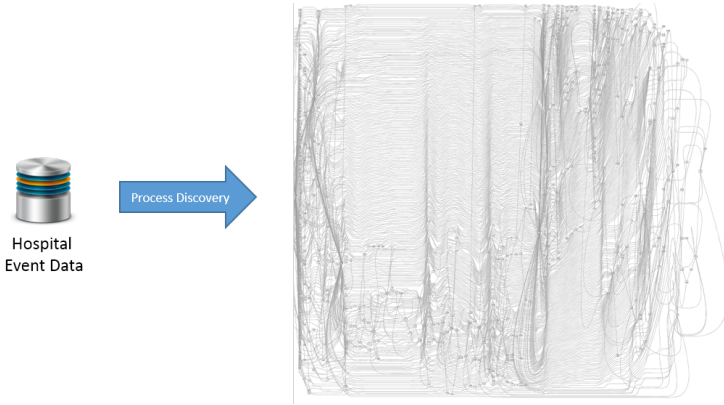


Figure 1.8: Example of a "spaghetti" process model obtained directly from the event log of a Dutch hospital.

This process model describes the control-flow of the process related to the diagnosis and treatment of over 1100 oncologic gynecology patients, described by over 150.000 events. There are over 600 different activities (e.g., tests, consultations) in the process.

The degree of control-flow structure in a process is usually related to the

number of different activities in the process and the relations between them. A common approach used in process mining to deal with control-flow variability is to filter infrequent behavior by removing the infrequent activities and paths, so only the most frequent ones are kept, hence a simpler model is obtained. This can indeed allow the analyst to get a clearer picture of what the process looks like in some cases. However, this has three main drawbacks. The first one is that exceptions could be critical to the process and are also interesting to analyze. If infrequent activities (related to exceptions) are filtered out, this information is lost. The second one is related to the fact that unstructured processes may have no "frequent" behavior. Therefore, threshold-based frequency filtering may remove or keep activities with relatively similar frequencies. The third one is that there is no standard guideline about how much should be filtered out. As an extreme example, the hospital log could be filtered out until only one activity remains (i.e., the most frequent). This will result in a process model that is as simple as it is incorrect.

Other process discovery techniques like the fuzzy miner [64] can deal with variability by clustering regions of activities with a lot of variability. They do present a simpler high-level model as a result, but only when a cluster is "expanded", all the hidden variability appears.

This thesis proposes a different approach: to consider that variability comes from different *variants* (i.e., versions of the process) grouped together, and that the process can be split in some way into such variants. Figure 1.9 illustrates this idea in the hospital example mentioned above: one could use derived or given dimensions present in the data (such as the "Diagnosis") to split the event data based on its different values using a process cube. The choice of which dimensions to be used normally depends on the available domain knowledge of the process. However, when such domain knowledge is missing, process variant detection techniques can be used to identify which dimensions (given or derived) can be used to maximize the exposure of variability in the data.

In the hospital example, splitting the data using the "Diagnosis" dimension results in a set of simpler process models (in terms of control-flow) compared to a model that includes all types of patients together (see Figure 1.8), as patients with similar diagnostics take similar exams and treatments.

The advantage of extracting variants of a process is that they (and the process models that represent them) tend to be simpler, and they can be compared with each other. Naturally, not all process variants are always *relevant* for the analysis purpose. Differences and similarities between relevant variants can uncover many useful insights and lead to a better understanding of the whole process.

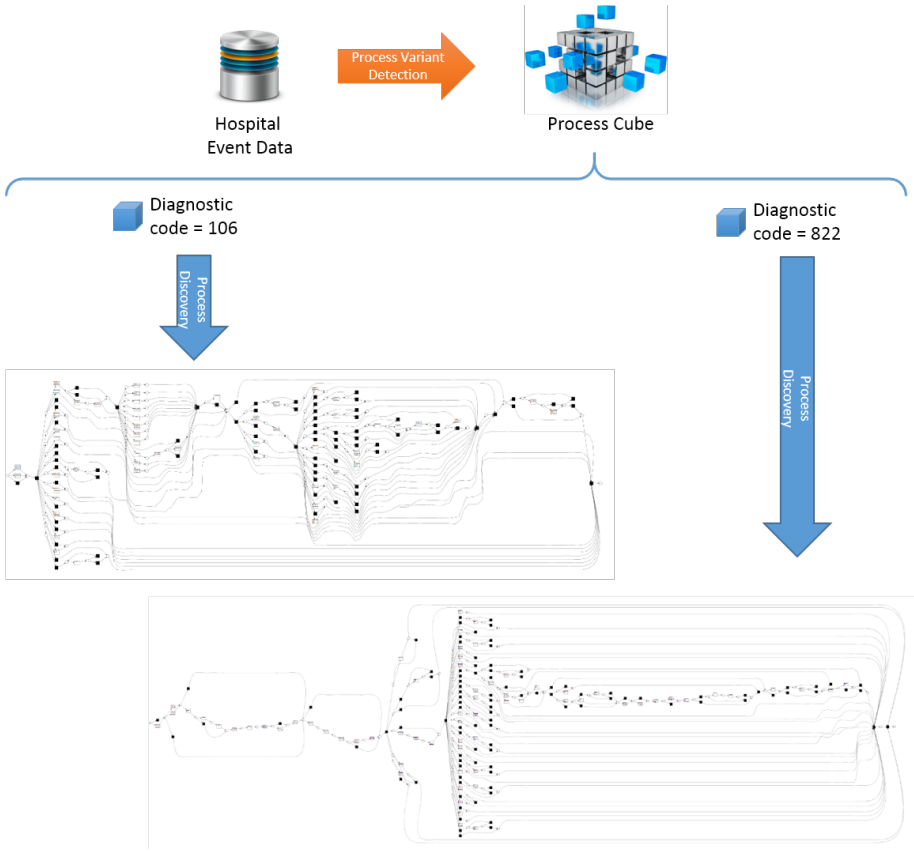


Figure 1.9: Example of an event log of a Dutch hospital being split into process variants based on the initial diagnostic of patients.

Figure 1.10 shows an example of a process model (i.e., a transition system) enhanced with case frequency information that compares the event data of patients with two different Diagnostic codes (i.e. 106 and 821) in terms of control-flow. In such a model, the main differences are highlighted. This will be discussed in much more detail further ahead.

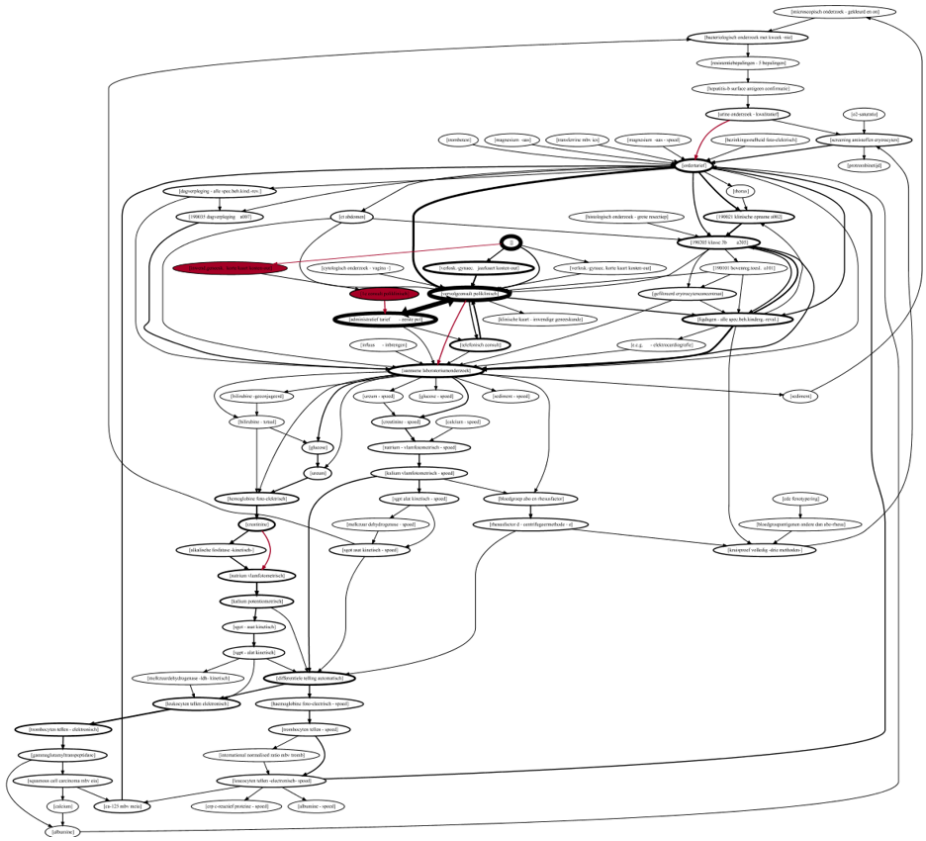


Figure 1.10: Control-flow comparison of patients with Diagnostic code = 106 and patients with Diagnostic code = 821. Colored states and arcs represent statistically significant differences in terms of frequency of occurrence.

Performance Variability

Processes with a relatively structured control-flow can still have a high variability in other process perspectives such as *performance*. However, performance is usually not considered by process discovery algorithms, hence the resulting process models only care about the control-flow structure of the process. To overcome this, process enhancement techniques are used to annotate process models with performance information, allowing analysts to visually inspect the

performance of the process in the process models and can identify bottlenecks based on such information [156].

From all the possible performance annotations, the most commonly used by process enhancement techniques are the average *duration* of activities, the time in-between activities and the average elapsed time in a case. Even though using averages can simplify analysis (i.e., fewer numbers to focus on), it has a drawback: single values (e.g., average, median) hide the underlying distribution. Averaging is useful only when the underlying distribution is (close to) either uniform or normal with a small standard deviation. In such cases, the average can be representative. Other types of distributions (e.g., skewed, exponential/poisson, normal with a large standard deviation) cannot be properly summarized by an average. In real life, performance is usually far from being constant or uniform, as the process is executed in different moments of time by different resources and often handling different case complexities (e.g., complex cases may take longer than simple cases). Many non-process-mining-related classical data analysis tools (e.g., SAS, SPSS) can be used to analyze such distributions.

However, since we are dealing with event data, performance analysis should not be done in isolation, but always in relation to the process in order to e.g., identify performance bottlenecks or points of performance improvement in the process.

Figure 1.11 illustrates the idea. In a similar fashion as Figure 1.10, it shows an example of an enhanced process model (i.e., a transition system) that compares the event data of patients with the same two different Diagnostic codes (i.e. 106 and 821), but this time in terms of performance. In such a model, the main differences are highlighted so that it is easy to detect performance differences in specific parts of the process. This will also be discussed in much more detail further ahead.

Differences in performance can be related to many factors e.g., control-flow, resource, etc. In the hospital process, performance could be affected by the complexity the treatment, the accuracy of the initial diagnostics, etc. Moreover, the actual grouping criteria is also a determinant factor for observing performance differences. In the hospital process, patients were simply grouped according to the initial diagnostics. However, in large and complex processes, defining a good grouping criteria is far from trivial. Most of the time is done manually and it is usually obtained through either trial-and-error or the use of domain knowledge, which is not always available.

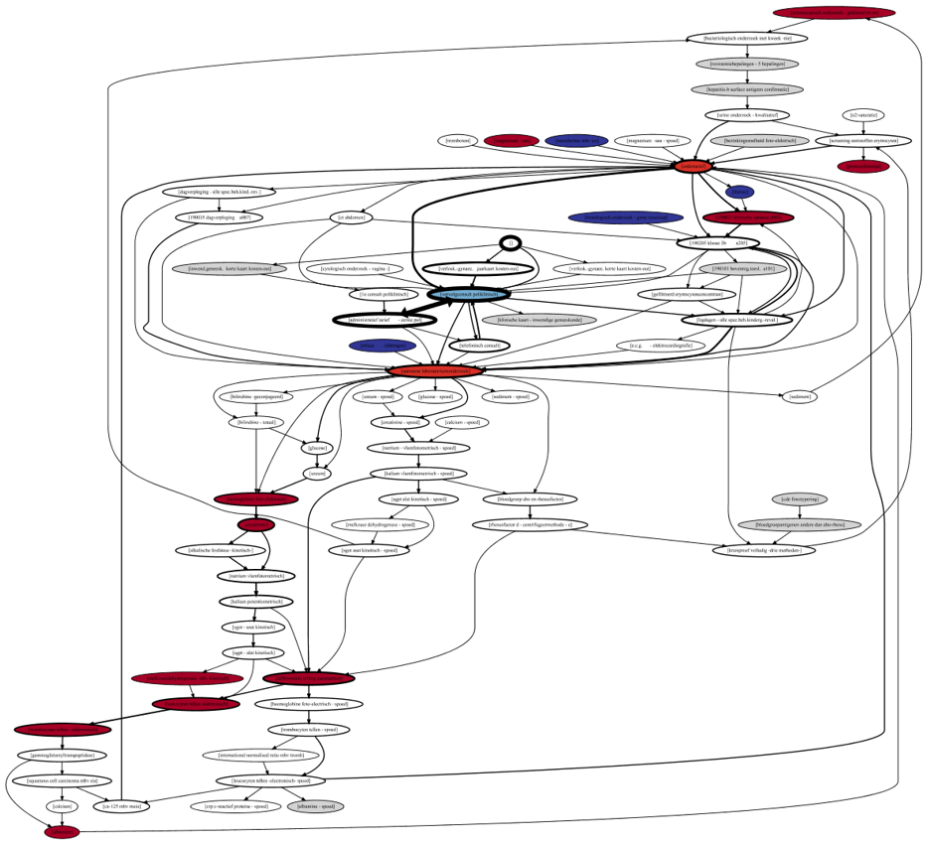


Figure 1.11: Performance comparison of patients with Diagnostic code = 106 and patients with Diagnostic code = 821. Colored states and arcs represent statistically significant differences in terms of elapsed time.

Resource & Data Variability

Control-flow process models can also be enhanced using other data attributes of events such as the resource that executes an activity, the customer age and income, etc. The executions of a process can be clustered into subgroups depending on resource and data attributes. Such subgroups might have different control-flow or performance over the process. This type of clustering enables

many types of analysis, such as understanding the differences in the *customer journey* for different type of customers.

In the hospital process, for example, all the patients treated by the same doctor, or that had similar exam results can be grouped together. It might be the case that one specific doctor has a much higher treatment success rate than the others, and it would be interesting to analyze and understand the reasons why this happens. However, these insights are not visible if the right perspectives are not included.

Most process mining techniques ignore this type of variability, as they mainly focus on control-flow, sometimes extending the scope to performance.

1.3 Opportunities for Tool Support in Process Mining

Hundreds of process mining techniques are available and their value has been proven in many case studies. See for example the twenty *case studies* on the webpage of the IEEE Task Force on Process Mining [72]. The open-source process mining framework *ProM* [172] provides over 1500 plug-ins and has been downloaded over 150.000 times. The growing number of commercial *process mining tools* (nowadays there are over 30 different vendors) further illustrate the uptake of process mining. Nevertheless, current tool support for process mining presents many opportunities for improvement. In this thesis, we will focus on three main opportunities, described as follows.

The first one is that process data often come from different and heterogeneous sources. Extracting process data from IT systems is not trivial. Moreover, most process event logs are obtained through manual extraction and preprocessing step. Such manual steps are often difficult to replicate. This is referred to in the process mining manifesto [152] as the challenge "*Finding, Merging, and Cleaning Event Data*". Extract, transform & load (ETL) techniques can be used to support the data extraction and preprocessing from several sources [180].

The second one is that process mining can be combined with other types of analysis. Process mining techniques have been proven to be useful in many applications, but their full potential is only unleashed when they are combined with other types of data analysis. This is referred to in the process mining manifesto [152] as the challenge "*Combining Process Mining With Other Types of Analysis*". Other disciplines such as complex event processing, sequence mining, pattern mining, natural language processing, simulation, machine learning, and

many others, can also contribute to achieving this goal.

The third one is that large-scale experimentation is not supported by existing process mining tools. Experiments in process mining rarely consist of a single application of a technique. Often, many techniques need to be chained together in a specific order. For example, a model is discovered and then it is checked for conformance with respect to the data. Moreover, the same chain of techniques can be executed hundreds or thousands of times. For example, when finding the parameter value of the process discovery technique that results in the best conformance. As a result, the manual execution of these chains of techniques can become tedious and error-prone, which jeopardize the repeatability and provenance of the experiments. These chains of techniques can be used for many purposes, such as the challenge "*Creating Representative Benchmarks*" in the process mining manifesto [152]. Such benchmarks should rely on statistical hypothesis testing, which often require large sample sizes.

This thesis proposes a new approach to capture these opportunities by combining process mining techniques with existing analytic workflow tools. Figure 1.12 illustrates the idea. This workflow can be used to extract, transform & load (ETL) the data in early steps and then perform several process mining

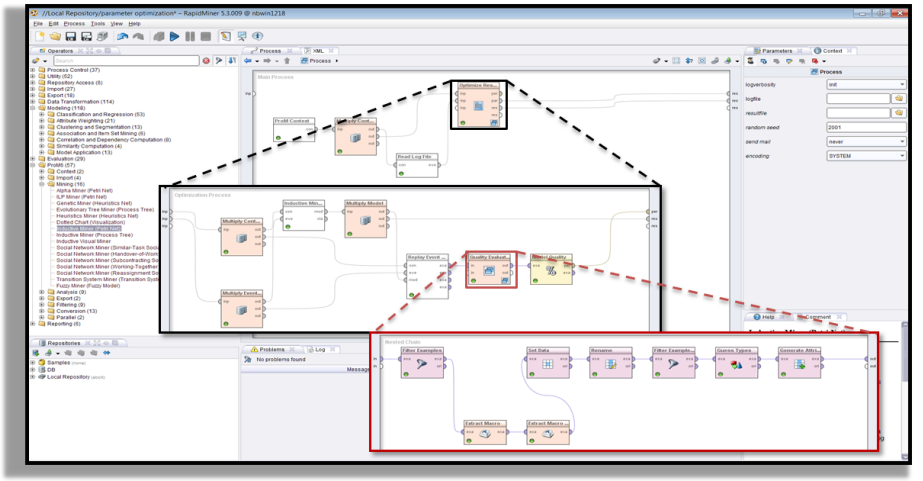


Figure 1.12: Example of an analytic workflow that combines ETL, process mining and non-process mining analysis steps and performs large-scale experimentation.

analysis combined with an arbitrary set of non-process mining analysis for any given number of process variants in any scale needed. Such type of workflows enable large-scale experimentation (including representative benchmarks) combining process mining and non-process-mining techniques in a clear, error-safe, reproducible, and transparent manner.

1.4 Contributions in this Thesis

Based on the two discussions presented above, several contributions are included in this thesis.

To address the challenges discussed in Section 1.2: "Dealing with Variability in Processes", the following three *technical* contributions to the body of scientific knowledge (CBSKs) are proposed.

Process Cubes: *A technique to support the interactive and consistent exploration of process variants* (Chapter 3). Process Cubes are the result of adapting OLAP-operations to explore process data, where each cell in a process cube contains events that can be converted into a process variant. Process cubes enable the consistent exploration of process variants, which can be used by other process mining techniques.

Process Variant Comparison: *A technique to compare process variants.* (Chapter 5). This technique is able to compare process variants in terms of behavior (using event dimensions) and in terms of business rules, based on their event logs. The results are projected into a process model that serves as a "map" in which the differences are clearly identified and can be pinpointed to specific parts (e.g., activities) of the process.

Process Variant Detection: *A technique to detect relevant process variants in a general setting.* (Chapter 6). This technique is able to detect process variants in process data by splitting cases based on the data attributes of their events. It uses statistical testing and unbiased variable selection to detect only relevant process variants. The result is a summary of relevant splittings, where each splitting leads to a set of variants. Each variant is then encoded into its corresponding set of traces. As a result, an enriched event log can be used by a process cube to split the event data into such variants using variant-related dimensions.

To address the challenges discussed in Section 1.3:"Opportunities for Tool

Support in Process Mining", the following *tooling* contributions to the body of scientific knowledge are proposed.

Process Mining Workflows: *Support the execution of process mining workflows.* (Chapter 4). The concept of a process mining workflow as a chain of process mining (and/or non-process mining) analysis steps is introduced. Process mining workflows can be used to completely describe arbitrary process mining experiments. Therefore, it enables full reproducibility of the results. The tool supporting these workflows is introduced and it is applied in several use cases.

Benchmark Frameworks: *Develop replicable and sound benchmarks.* (Chapters 7 and 8). Process mining workflows are used to define two frameworks: one for benchmarking process discovery techniques and the other for benchmarking concept drift detection techniques. These frameworks are not meant only for comparing techniques: they also allow for benchmarking techniques at a statistical level for specific process characteristics, or to perform parameter sensitivity analysis. For example, the effect of parallelism on the quality of models produced by a process discovery technique can be studied.

Additionally, this tesis adds three *empirical* contributions to the body of scientific knowledge in the form of case studies.

Case Studies: (Chapters 9, 10, and 11). Three case studies using real-life event data show the relevance of the techniques presented in this thesis through their application in solving problems related to different organizations and industries, such as: Telecommunications, Printing & Digitalization, and Education.

1.5 Thesis Structure

This thesis is composed of twelve chapters that are organized into five parts. Figure 1.13 shows an overview of the parts of this thesis and the chapters included in them.

Part I serves as an introduction and motivation for this thesis. Next to this introductory chapter, **Chapter 2: Preliminaries** introduces the basic concepts and notations used in the remainder of this thesis.

Part II describes the main contributions proposed in this thesis. It includes the following chapters:

Chapter 3: Process Cubes. This chapter introduces the concept of event

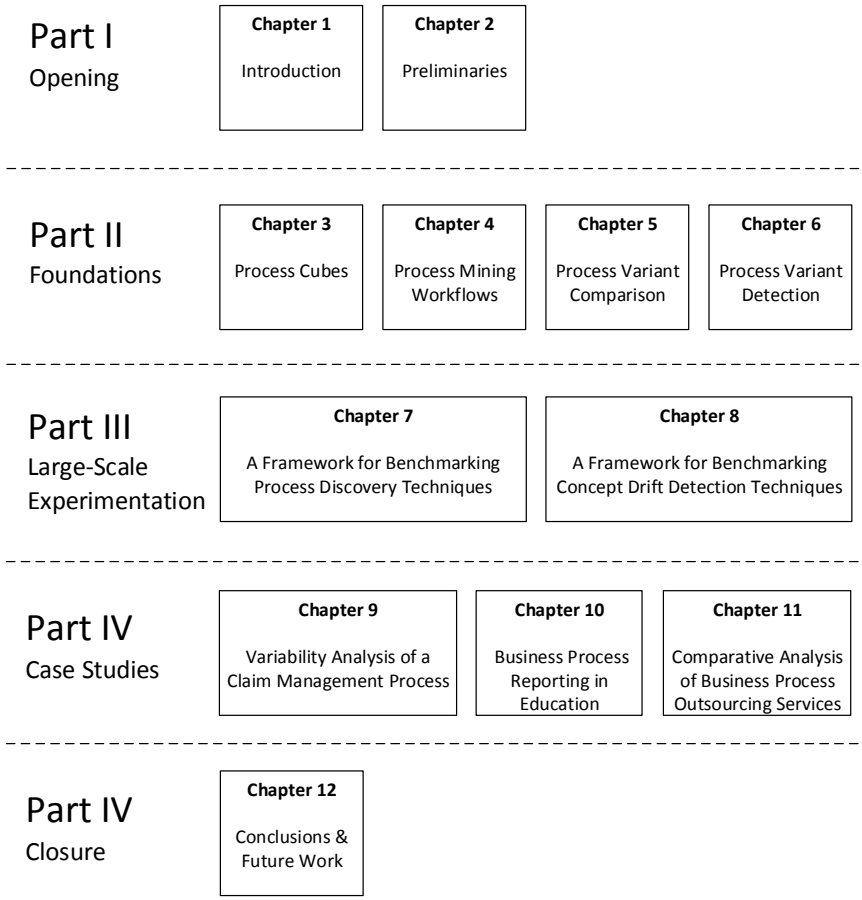


Figure 1.13: Structure of this thesis. The twelve chapters are organized into five parts.

dimensions, and describes how OLAP-cube operations can be adapted for event data, and describes how they can be used to perform multidimensional exploration of processes. Finally, the natural connection of process cubes with process comparison and the execution of process mining workflows from the cells of the cube is discussed. This chapter is based on the work presented in [16].

Chapter 4: Process Mining Workflows. This chapter introduces the concept of process mining workflows, and proposes several process mining workflow patterns i.e. (use cases) using building blocks. It also describes how process mining workflows can interact with other types of analysis and with the process cubes introduced in the previous chapter. This chapter is based on the work presented in [11, 169].

Chapter 5: Process Variant Comparison. This chapter introduces a technique to compare process variants in terms of behavior (using event dimensions) and in terms of business rules. This chapter is based on the work initially presented in [12] and later extended in [13].

Chapter 6: Process Variant Detection. This chapter introduces the concept of process variants, and proposes a technique to automatically detect them using event attributes. This chapter is based on the work presented in [17].

Part III presents two applications of the process mining workflows presented in Chapter 4 to propose frameworks for benchmarking process mining techniques by leveraging on the large-scale experimentation capabilities enabled by the use of such process mining workflows. The chapters included in this part are:

Chapter 7: A Framework for Benchmarking Process Discovery Techniques. This chapter presents a framework for comparing process discovery techniques in different scenarios for different populations of processes. This chapter is based on the work presented in [85].

Chapter 8: A Framework for Benchmarking Concept Drift Detection Techniques. This chapter presents a framework for comparing, in the scope of event data, concept drift detection techniques in different scenarios for different populations of processes.

Part IV describe the empirical application of the techniques proposed in Part II in three case studies using real-life data sets. Table 1.2 summarizes the usage of these techniques in each case study.

Table 1.2: Usage of the techniques presented in Part II in each case study.

	Process Cubes	Process Mining Workflows	Process Comparison	Process Variant Detection
Chapter 9	X		X	X
Chapter 10	X	X		
Chapter 11		X	X	

Chapter 9: Variability Analysis of a Claim Management Process. This chapter describes the application of process variant detection, process cubes, and process comparison techniques to analyze and compare process variants in the claim management process of a large telecommunications company.

Chapter 10: Business Process Reporting in Education. This chapter describes the application of process cubes and process mining workflow techniques to automate the generation of reports obtained from educational data in a large scale, combining learning analytics and process mining techniques.

Chapter 11: Comparative Analysis of Business Process Outsourcing Services. This chapter describes the use of process mining workflows and process comparison techniques to analyze and understand the variability of a document digitalization service process in a business process outsourcing organization.

Finally, Part V concludes this thesis by summarizing the contributions presented on this thesis, discussing their limitations and proposing future research directions in **Chapter 12: Conclusions**.

Chapter 2

Preliminaries

This chapter introduces the existing concepts and notations that will be used throughout this thesis. It is organized as follows. Section 2.1 introduces basic mathematical concepts and notations such as sets and functions. Section 2.2 provides a formal definition of event logs and their components, and discusses existing standards. Section 2.3 briefly describes well-known process modeling notations and provides a formal definition of transition systems as process models, defining how they can be constructed using event data. Finally, Section 2.4 describes the running example that will be used throughout this thesis.

2.1 Basic Notations

Definition 2.1 (Set)**.** *A set is an unordered collection of distinct objects of any nature.*

The symbol $\in$ is used to denote the membership of an object in a set, and its negation ($\notin$) is used to denote the opposite. For example, given a set X, the expression $a \in X$ means that a is an element of X, while the expression $b \notin X$ means that b is not an element of X.

There are two ways to define a set: by intension or by extension. An *intensional definition* uses a rule or semantic description, e.g., "the colors of the Dutch flag". An *extensional definition* lists each element of the set in curly brackets, e.g., {`red`, `white`, `blue`}.

Some specific infinite sets of elements used in this thesis are also commonly used in mathematics. They are considered as "standard" sets.

Definition 2.2 (Standard Sets of Elements)**.** *Standard sets of elements are described as follows:*

- $\mathbb{N}$ *denotes the set of natural numbers (*$\mathbb{N}^0$ *includes 0).*
- $\mathbb{Z}$ *denotes the set of integer numbers (*$\mathbb{Z}^+$ *denotes positive integers).*
- $\mathbb{R}$ *denotes the set of real numbers.*

For sets with a large number of elements, an extensional definition is sometimes inconvenient. Instead of listing all the elements of a set, an abbreviated definition can be made using the set-builder notation of the form {*variable*|*conditions*} which defines a set with all the values of the *variable* for which the *conditions* hold (i.e., are true). For example, in the expression $\{a \in \mathbb{Z} | \exists b \in \mathbb{Z} : a = 2 * b\}$, the variable a is an integer and the condition states that there is an integer b for which a is the double. Note that this expression simply defines the set of even integer numbers.

Let X and Y be two sets. The standard notation for operations over sets used in this thesis is defined as follows:

- $\emptyset = \{\ \}$ denotes the *empty set*.
- $|X|$ denotes the *cardinality* (i.e., number of elements) of the set X.
- $X \cup Y = \{a | a \in X \vee a \in Y\}$ denotes the *union* of X and Y.
- $X \cap Y = \{a | a \in X \wedge a \in Y\}$ denotes the *intersection* of X and Y.
- $X \setminus Y = \{a \in X | a \notin Y\}$ denotes the *difference* of X and Y.
- $X \subseteq Y \Leftrightarrow |X \cap Y| = |X|$ denotes that X is a *subset* of Y, i.e., every element of X is also an element of Y.
- $X \subset Y \Leftrightarrow X \subseteq Y \wedge |X| < |Y|$ denotes that X is a *strict subset* of Y.
- $\mathbb{P}(X) = \{Y | Y \subseteq X\}$ denotes the *power set* (i.e., the set of all subsets) of X.
- $X \times Y = \{(x, y) | x \in X \wedge y \in Y\}$ denotes the *cartesian product* of X and Y, where (x, y) is an ordered pair (i.e., a *tuple*).

- $X^n = X_1 \times X_2 \times ... \times X_n$ where $X = X_1 = X_2 = ... = X_n$, is the *n-fold cartesian product* of X. All the elements in X^n are n-tuples of size n.

Definition 2.3 (Relation). *Let $X_1, ..., X_n$ be sets. A n-ary relation $R \subseteq X_1 \times ... \times X_n$ defines a mapping between elements of these sets. This mapping is observed in the elements of the relation. Any n-tuple $(x_1, ..., x_n) \in R$ where $x_1 \in X_1, ..., x_n \in X_n$ relates the values of $x_1, ..., x_n$ to each other.*

Functions and partial functions are a special case of a relation between two sets, and they map elements of one set to elements of another set.

Definition 2.4 (Partial Function). *A partial function $pf : X \nrightarrow Y$, is a subset of the cartesian product of X and Y, where each element in X is related to at most one element of Y:*

$$\forall_{x \in X} \forall_{y_1 \in Y} \forall_{y_2 \in Y} \big((x, y_1) \in pf \land (x, y_2) \in pf \big) \Rightarrow y_1 = y_2$$

Note that a partial function $X \nrightarrow Y$ allows elements of X to not be mapped to any element of Y. Total functions enforce a mapping of all elements of X to exactly one element of Y.

Definition 2.5 (Total Function). *A function $f : X \rightarrow Y$, is also a subset of the cartesian product of X and Y, where each element in X* is always *related to exactly one element of Y:*

$$\forall_{x \in X} \forall_{y_1 \in Y} \forall_{y_2 \in Y} \big((x, y_1) \in f \land (x, y_2) \in f \big) \Rightarrow y_1 = y_2 \land \forall_{x \in X} \exists_{y \in Y} \big((x, y) \in f \big)$$

For any function or partial function $f : X \nrightarrow Y$, the domain of f is denoted as $dom(f) = \{x \in X | \exists_{y \in Y} (x, y) \in f\}$, and the range of f is denoted as $rng(f) = \{y \in Y | \exists_{x \in X} (x, y) \in f\}$. Additionally, for any function or partial function f, a relation $(x, y) \in f$ can alternatively be denoted as $f(x) = y$.

A function $f : X \rightarrow Y$ is *surjective* if each element of Y is related to an element of X: $\forall_{y \in Y} \exists_{x \in X}\ f(x) = y$. A partial function $f : X \nrightarrow Y$ is *injective* if each element of X is related to a different element of Y: $\forall_{x_1 \in X} \forall_{x_2 \in X}\ \big(f(x_1) = f(x_2) \big) \Rightarrow x_1 = x_2$. A function is *bijective* if it is surjective and injective.

A *multiset* is an unordered collection of objects, where the objects can be present multiple times in the collection.

Definition 2.6 (Multiset). *Given a set X, a multiset M over X is defined as the function $M : X \rightarrow \mathbb{N}_0$.*

A multiset can be defined by extension using square brackets and superindices to indicate the multiplicity of elements, where an element $x \in X$ is represented as $[x^{M(x)}]$. For example, $M = [a, b^2]$ contains one a and two b's.

The set of all the possible multisets over X is defined as $\mathbb{B}(X)$. The *size* of a multiset corresponds to the number of distinct elements on it, regardless of their cardinality.

The *sum* of two multisets M_1 and M_2 over the set X, i.e., $M_1 \uplus M_2$ yields a resulting multiset M' with $M'(x) = M_1(x) + M_2(x)$.

As a consequence, a set X can be seen as a multiset where the multiplicity of all its elements is one.

A *sequence* is an ordered collection of objects, where the objects can be present multiple times in the sequence.

Definition 2.7 (Sequence)**.** *Let X be a set. A sequence of length $n \in \mathbb{N}$ over the elements of X is defined as the function $s \in \{1, 2, ..., n\} \to X$, which defines the order in which elements appear in the sequence. Equivalently, a sequence of length $n \in \mathbb{N}$ over the elements of X can be defined as a n-tuple $s = \langle s_1, s_2, ..., s_n \rangle \in X^n$. The i-th element of the sequence s, for any $1 \leq i \leq n$ is defined as $s(i) \in X$, denoted as simply s_i.*

Given a set X, the set of all the sequences of all possible lengths over X is denoted as $X^* = \bigcup_{i \in \mathbb{N}} X^i$.

Now that all the necessary basic mathematical concepts have been introduced, we proceed to introduce the process-mining-related concepts that will be used throughout this thesis.

2.2 Events as Observed Executions of a Process

Using the notation and definitions presented previously, several *universes* are defined. The universes mentioned and used throughout this thesis are described as follows.

Definition 2.8 (Universes)**.** *Universes are infinite sets of elements:*

- *$\mathcal{V}$ is the universe of* values*, including numbers, characters, names, etc.*
- *$\mathcal{E}$ is the universe of* events*.*
- *$\mathcal{N}$ is the universe of* attribute names*.*

As mentioned previously in Section 1.1, information systems can record the execution of a process. Every time that something "happens" in the process, an event $e \in \mathcal{E}$ is recorded. Events are unique and they can be characterized by *attributes*.

Definition 2.9 (Attribute)**.** *Attributes can relate events to values through the function $\# : \mathcal{N} \rightarrow (\mathcal{E} \nrightarrow \mathcal{V})$. For any attribute $a \in \mathcal{N}$, the partial function $\#(a)$ can relate events to values of the attribute a. In the remainder, function $\#(a)$ is shortened as $\#_a$.*

For an attribute $a \in \mathcal{N}$ and an event $e \in \mathcal{E}$, if $e \in dom(\#_a)$, then the event e has a value for the attribute a, indicated as $\#_a(e) = v \in \mathcal{V}$. If $e \notin dom(\#_a)$, then the event e does not have a value for the attribute a. We write $\#_a(e) = \perp$ to indicate this.

Events can be related to multiple attributes such as costs, resources, customers, purchase amounts, etc. However, there are three specific event attributes that are almost always present in an event log:

- Case ID: describes the specific *case* related to this event, so that an event can be related to other events with the same case id (denoted as $\#_{case\_id}$),
- Activity: describes the specific *activity* executed in an event (denoted as $\#_{activity}$),
- Timestamp describes the moment in time when an event was executed (denoted as $\#_{time}$).

For example, in Figure 1.4, the first event e (leftmost) of the first trace (i.e., article #3025) is characterized by the three following attributes: $\#_{case\_id}(e) = 3025$, $\#_{activity}(e) =$ `invite reviewers`, and $\#_{time}(e) =$ 01/08/2017.

An event can be related to a set of attributes for which it has a value, by the function $atts : \mathcal{E} \rightarrow \mathbb{P}(\mathcal{N})$, where for any event $e \in \mathcal{E}$, $atts(e)$ is defined as $\{a \in \mathcal{N} | e \in dom(\#_a)\}$.

As mentioned before, events can be grouped together if they refer to the same *case* of the process. For example, they may refer to the same patient in a hospital, or to the same production order in a factory. A *trace* records the execution of a case of a process.

Definition 2.10 (Trace)**.** *A trace (i.e., case) $\sigma \in \mathcal{E}^*$ is a finite sequence of events, where for any $e_1, e_2 \in \sigma : \#_{case\_id}(e_1) = \#_{case\_id}(e_2)$. In other words, all the events in a trace have the same value for the case id attribute.*

The length of a trace is denoted as $|\sigma|$. The k^{th} event of a trace is denoted as $\sigma(k)$ with $k \leq |\sigma|$. The last event of a trace is denoted as $\sigma(|\sigma|)$. The prefix of a trace containing its first k events is defined by the function $pref^k \subseteq \mathcal{E}^* \rightarrow \mathcal{E}^*$, with $k \leq |\sigma|$. Note that $pref^0(\sigma) = \langle\rangle$. The set of all the prefixes of a trace σ is defined as $pref^\diamond(\sigma) = \bigcup_{k=0}^{|\sigma|}\{pref^k(\sigma)\}$.

Traces can also be manipulated in order to extract process characteristics and include them as attributes in the events of the trace, as proposed in [45].

Definition 2.11 (Trace Manipulation Function)**.** *A trace manipulation function is defined as $\mathcal{T} : \mathcal{E}^* \rightarrow \mathcal{E}^*$. This function is defined for any trace $\sigma = \langle e_1, ..., e_n\rangle \in \mathcal{E}^*$ as $\mathcal{T}(\sigma) = \langle f_1, ..., f_n\rangle \in \mathcal{E}^*$ where all $e \in \sigma$ and all $f \in \mathcal{T}(\sigma)$ are unique events, and $|\sigma| = |\mathcal{T}(\sigma)|$, such that for any attribute $a \in \mathcal{N}$ and $\forall_{1\leq i\leq n} : e_i \in dom(\#_a) \Rightarrow \#_a(e_i) = \#_a(f_i)$.*

A trace manipulation function creates new events by copying all the attribute values of existing events and adding extra attributes. It does not change the value of event attributes when such attributes already have a value in the original trace, and it does not add or remove events.

There are several ways to manipulate a trace by adding extra event attributes. These extra attributes can be related to different perspectives such as control-flow, performance, resource, and costs, but also to the context of the process and variant information. For example, process variants found by the approach presented in Chapter 6 can be used to enrich the original event data, e.g., by encoding the variant-related information as new additional attributes.

The work presented in [45] discusses a list of over twenty different trace manipulation functions. From this list, we used four manipulation functions to extend all the event logs used in this thesis. A brief description of them is provided as follows.

Next Activity (CFP2): each event is annotated with the name of the activity executed afterwards for the same trace. Formally, this is defined as: $\mathcal{T}_{CFP2}(\langle e_1, \ldots, e_n\rangle) = \langle f_1, \ldots, f_n\rangle$ such that $\forall_{1\leq i<n} : \#_{next\_activity}(f_i) = \#_{activity}(e_{i+1})$, with $\#_{next\_activity}(e_n) = \perp$.

Duration (TIP1): each event is annotated with the time difference between the current activity and the previous activity of the same case. Formally, this is defined as: $\mathcal{T}_{TIP1}(\langle e_1, \ldots, e_n\rangle) = \langle f_1, \ldots, f_n\rangle$ such that $\forall_{1<i\leq n}$: $\#_{duration}(f_i) = \#_{time}(e_i) - \#_{time}(e_{i-1})$, with $\#_{duration}(f_1) = \perp$.

Elapsed Time (TIP2): each event is annotated with the time difference between the current activity and the first activity of the same case. For-

mally, this is defined as: $\mathcal{T}_{TIP2}(\langle e_1, \ldots, e_n \rangle) = \langle f_1, \ldots, f_n \rangle$ such that $\forall_{1 \leq i \leq n} : \#_{elapsed}(f_i) = \#_{time}(e_i) - \#_{time}(e_1)$.

Attribute Cascading (DFP2) : each event is annotated with the latest value of every available attribute contained in previous events of the same trace. Formally, this is defined as: $\mathcal{T}_{DFP2}(\langle e_1, \ldots, e_n \rangle) = \langle f_1, \ldots, f_n \rangle$ such that $\forall_{a \in \mathcal{N}} : \#_a(f_1) = \#_a(e_1)$, and $\forall_{1 < i \leq n} : e_i \in dom(\#_a) \Rightarrow \#_a(f_i) = \#_a(e_i)$, and $e_i \notin dom(\#_a) \wedge f_{i-1} \in dom(\#_a) \Rightarrow \#_a(f_i) = \#_a(f_{i-1})$.

The brackets in each manipulation function contains the special attribute name as they have been defined in [45].

A collection of traces (manipulated or not) is defined as an *event log*.

Definition 2.12 (Event Log)**.** *An event log $L \subseteq \mathcal{E}^*$ is a finite set of traces such that given any two traces $\sigma_1, \sigma_2 \in L : \exists_e : e \in \sigma_1 \wedge e \in \sigma_2 \Rightarrow \sigma_1 = \sigma_2$.*

Within an event log, each event is unique and can appear only once in one trace. The set of all the prefixes of traces of an event log L is defined as:

$$P_L = \bigcup_{\sigma \in L} pref^{\diamond}(\sigma)$$

Note that for any trace σ, $\langle\rangle$ and σ are also considered to be a prefix of σ. The set of all the events in an event log L is defined as:

$$E_L = \bigcup_{\sigma \in L} \{e \in \sigma\}$$

In this thesis, an event log is assumed to contain traces related to the same process. An event log may contain variations of the same process (i.e., *process variants*).

Definition 2.13 (Process Variant (Log))**.** *Given an event log L, a set of process variants of L is defined as $V_L \subseteq \mathbb{P}(L)$, such that $|V_L| > 1$ and that for any $v_1, v_2 \in V_L, v_1 \subseteq v_2 \Rightarrow v_1 = v_2$. Each process variant $v \in V_L$ is simply a subset of traces of L.*

Process variants are typically constructed to group together homogeneous behavior. A process variant refers to a *variation* or *version* of the process, hence there has to be least another *version* of the process in the event log in order to call them as *process variants*. In this thesis, process variants are considered as non dominant, i.e., a variant cannot fully contain another variant. Note that

traces in the event log are allowed to be in more than one variant. For example, if we distribute the cases of an event log into process variants depending on the year in which any event was executed, then cases that started in 2017 and finished in 2018 can be related to the process variants related to both years.

2.3 Process Modeling Notations

Process models use notations to represent some process perspectives (usually, the control-flow of the process). For process models, several notations exist in literature. Note that these notations have several properties and characteristics. The most commonly used in process mining are: Petri nets, process trees, BPMN and transition systems. The remainder of this section briefly describes the notations mentioned above. For a formal definition of these process modeling notations, the reader is referred to [156]. Since this thesis heavily uses transition systems, these will be formally described in Section 2.3.4.

2.3.1 Petri Nets

Petri nets are a well known process modeling language and are one of the first to provide explicit support for concurrency. Petri nets are bipartite graphs containing places, transitions, and directed arcs between them. Places can contain *tokens* that can flow through transitions, and the state of all places (i.e. a *marking*) represents the state of the process execution.

In the context of process mining, not all types of Petri nets are used. Workflow nets (WF-nets) are a subset of Petri nets that is specially adapted to represent the workflow of process activities. One of the main particularities of WF-nets are the presence of a *source*, i.e., a place with no incoming arcs, and a *sink*, i.e., a place with no outgoing arcs. Another particularity of WF-nets is the presence of initial and final *markings* which define the initial and final desired values for all the places in the net.

WF-nets also have clear execution semantics. Because of this, they can be mapped onto transition systems, e.g., by building a so-called *reachability graph*. WF-nets are mainly used to represent the control-flow perspective of a process, as they have difficulties capturing data and performance-related aspects. This issue is partially addressed by *Colored Petri Nets* (CPN) [80] and *Data Petri nets* (DPN) [156] in which tokens carry such data aspects (i.e., case level). WF-nets can contain *deadlocks* (i.e., non-final markings with no outgoing transitions),

livelocks (i.e., transitions are enabled, but it is impossible to reach the final marking), therefore, *soundness* is not guaranteed.

Figure 2.1 shows a WF-net that represents the journal revision process (see Section 1.1). This WF-net was discovered from the event log shown in Figure 1.4 using the approach introduced in [167].

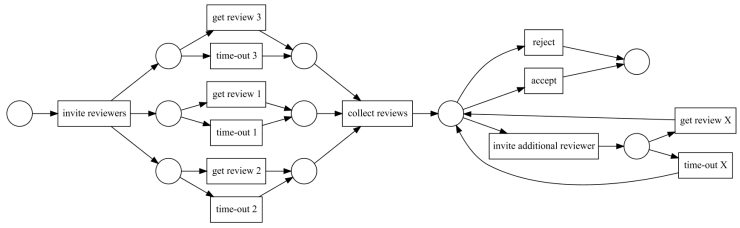


Figure 2.1: Petri net (WF-net) model representing the journal revision process.

2.3.2 Process Trees

A process tree is a hierarchical tree-structured process model where the inner nodes are operators (e.g., sequence, choice, parallel) and the leaves are activities [158]. Process trees are block-structured, and they guarantee soundness by design (i.e., no dead-or-live locks). They also have executable semantics, and they can be directly mapped onto *workflow nets* (i.e., a sub-class of Petri nets). Therefore, they can be transitively mapped onto transition systems.

Figure 2.2 shows a process tree that represents the simplified journal revision process (see Fig. 1.4). This process tree was discovered from the event log shown in Figure 1.4 using the approach introduced in [102].

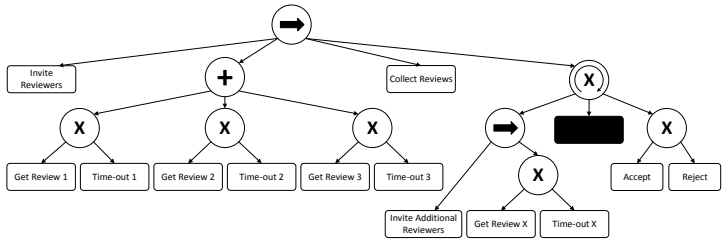


Figure 2.2: Process tree model representing the simplified journal revision process.

2.3.3 BPMN

The *business process model and notation* (BPMN) has become one of the most common notations to model processes. BPMN has been standardized by the OMG[1] in collaboration with several BPM software vendors and has been recently published by the ISO [2]. In its latest version, BPMN offers a large collection of constructs (over 50) that can be used to model specific behavioral settings.

BPMN has clear execution semantics (e.g., BPMN 2.0 can be directly executed in workflow engines). The mapping between Petri nets and BPMN is not completely bidirectional, but BPMN can be mapped to transition systems directly.

Figure 2.3 shows a BPMN model that represents the simplified journal revision process (see Fig. 1.4). This BPMN model was discovered from the event log shown in Figure 1.4 using the approach introduced in [39].

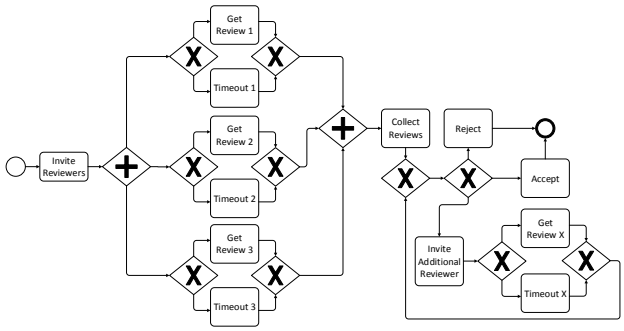


Figure 2.3: BPMN model representing the journal revision process.

2.3.4 Transition Systems

Transition systems are commonly used in computer science to describe the possible behavior of discrete systems. Since the execution of a process is also discrete (i.e., marked by events), transition systems can be used to represent the executions of a process. Transition systems are considered as the most basic representation of processes. This is because of the simplicity of this notation, com-

[1] `http://www.omg.org/spec/BPMN/`
[2] Resolution ID: ISO/IEC 19510:2013 (`https://www.iso.org/standard/62652.html`)

posed of only two constructs: *states* and *transitions* between them. Therefore, transition systems can easily represent the current state of a process execution.

Any process modeling notation with execution semantics can be mapped onto a transition system [156]. Moreover, transition systems are not limited to the control-flow perspective. They can be used, for example, to describe the interaction of resources in a process, i.e., social networks.

Based on the above, **transition systems are the process modeling notation that will be used throughout this thesis**.

As mentioned before, transition systems are composed of *states* and of *transitions* between them. A transition is defined by an activity being executed, triggering the current state of the process to move from a *source* state to a *target* state.

Let $\mathcal{R}^s$ be the universe of all the possible state representations. For example, in Figure 2.4, the state (i.e., `<Invite Reviewers>` $\in \mathcal{R}^s$) represents that the activity `Invite Reviewers` was executed.

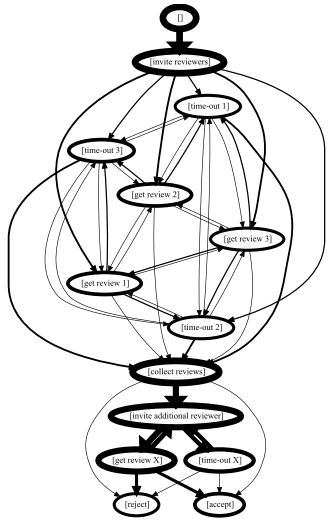


Figure 2.4: Transition system representing the journal revision process. Transition labels are hidden for improving readability.

Let $\mathcal{R}^a$ be the universe of all the possible activity representations. The activity representation of an event defines the type of activity that was executed.

This is commonly obtained from the event attributes and usually refers to

the `activity` event attribute. For example, in Figure 1.4, the first event of the first trace (i.e., article #3025) can be related to the activity `Invite Reviewers` $\in \mathcal{R}^a$.

Prefixes of traces can be mapped to states and transitions using representation functions that define how these prefixes are interpreted.

The *state representation* function is defined as $r^s : \mathcal{E}^* \rightarrow \mathcal{R}^s$. This function relates (prefixes of) traces to states in a transition system. Given an empty (prefix of a) trace, we denote the *empty state* as a special element $r^s(\langle\rangle) = \bot \in \mathcal{R}^s$

The *activity representation* function is defined as $r^a : \mathcal{E} \rightarrow \mathcal{R}^a$. This function relates events to activities.

When using a state representation function r^s and an activity representation function r^a together, (prefixes of) traces can be related to transitions in a transition system, as the activity and the source and target states of the transition can be identified using r^s and r^a. The set of all possible representations of transitions is defined as $\mathcal{R}^t \subseteq \mathcal{R}^s \times \mathcal{R}^a \times \mathcal{R}^s$. A transition $t \in \mathcal{R}^t$ is a triplet (s_1, a, s_2) where $s_1, s_2 \in \mathcal{R}^s$ are the source and target states and $a \in \mathcal{R}^a$ is the activity executed. Note that in this thesis only the `activity` event attribute is used to determine states and transitions. However, other event attributes can be used instead. For example, if a `resource` attribute is used in the state and activity representation functions, the resulting transition system will be a social network where states and transitions correspond to resources (e.g., employees) that execute events.

It is important to mention that transition systems do not inherently filter out infrequent behavior, as they represent *all* the behavior observed in an event log. In fact, they often over-approximate behavior (e.g., in the case of loops). If infrequent behavior needs to be filtered out, this must be done directly in the event data prior to the creation of the transition system or in a post-processing stage (see Chapter 5 for more details).

Now that we have defined the state and activity representation functions, we proceed to define transition systems.

Definition 2.14 (Transition System)**.** *Given an event log L, a state representation function r^s and an activity representation function r^a, a* transition system $TS^{(r^s, r^a, L)}$ *is defined as a triplet (S, A, T) where $S = \{r^s(\sigma) \mid \sigma \in P_L\}$ is the set of states, $A = \{r^a(e) \mid e \in E_L\}$ is the set of activities and $T = \{(s_1, a, s_2) \in S \times A \times S \mid \exists_{\sigma \in P_L \setminus \{\langle\rangle\}} : s_1 = r^s(pref^{|\sigma|-1}(\sigma)) \wedge a = r^a(\sigma(|\sigma|)) \wedge s_2 = r^s(\sigma)\}$ is the set of valid transitions between states.*

Given any trace prefix $\sigma = \langle e \rangle$ (i.e., $|\sigma| = 1$), then $pref^{|\sigma|-1}(\sigma) = \langle\rangle$ (see

Section 2.2). Hence, the only transition $t = (s_1, a, s_2)$ that can be defined for σ corresponds to $t = (r^s(\langle\rangle), r^a(e), r^s(e))$ which is the one that goes from the empty state $r^s(\langle\rangle)$ to the state $r^s(e)$. This empty state acts as the only *initial state* (i.e., the state with no incoming transitions) for the transition system, and all the other non-empty states in the transition system are (transitively) connected to it. However, the empty state can only be the *source* of a transition, but never the *target*. Therefore, the empty state cannot be a final state (i.e., a state with no outgoing transitions). In this thesis, we do not impose a single final state. Instead, any state with no outgoing transitions can be considered as a final state.

Also note that the structure of a transition system is affected by the state and activity representation functions used to create it.

As an example, Figure 2.4 showed the transition system that was learned based on the event log L related to the journal revision process introduced in Section 1.1 while using the state representation function defined as $r^s(\sigma) = \#_{activity}(\sigma(|\sigma|))$ for $\sigma \in P_L$ and the activity representation function $r^a(e) = \#_{activity}(e)$ for $e \in E_L$. Note that the special case of an empty trace is defined as: $\#_{activity}(\langle\rangle) = \{\ \}$. This transition system was discovered from this event log using the approach introduced in [162]. In this transition system, (prefixes of) traces are mapped into states and transitions as the activity name of their last event.

Alternatively, other state and activity representation functions can be used, which will result in structurally different transition systems. For example, Figure 2.5, shows a different representation for the same event log L. This transition system was created using the state representation function defined as

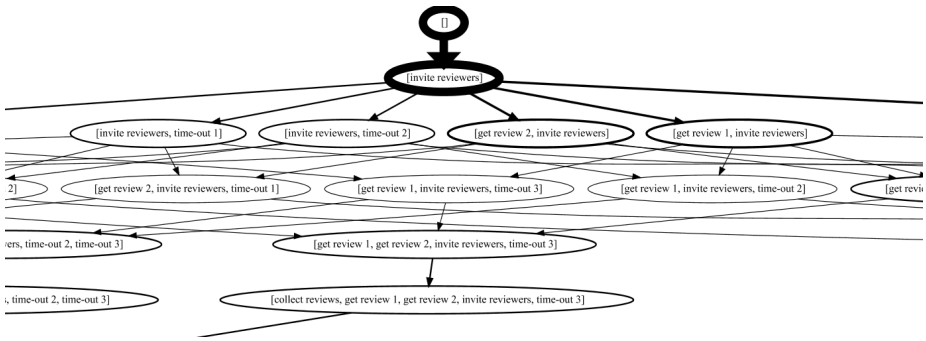


Figure 2.5: Fragment of an alternative transition system representing the journal revision process. The full transition system is not shown because of its size.

$r^s(\sigma) = \{\#_{activity}(e)|e \in \sigma\}$, for $\sigma \in P_L$ and the activity representation function $r^a(e) = \#_{activity}(e)$, for $e \in E_L$. In this transition system, (prefixes of) traces are mapped into states as the set of activity names of all their events, and into transitions as the activity name of their last event. For more information and detailed discussions on state and event representations in transition systems, the reader is referred to [162].

Now that the basic concepts have been introduced, we present the running example that will be used in chapters of Parts II i.e., Chapters 3 to 6.

2.4 Running Example: Road Fines

A *real-life* event log is used throughout this thesis as a running example. This event log describes the recorded execution of the *road fines management process* of a local police in Italy (publicly available in [43]). It contains 561.470 events for 150.370 road fines created between January 2000 and June 2013. Each fine relates to four events on average.

The list of attributes contained in this event log are presented in Table 2.1. From this list, the last three event attributes were generated using trace manipulation functions (see Definition 2.11).

Table 2.1: List of attributes that can be related to events of the *road fines* management process event log.

Event attribute	**Description**
Activity	The name of the task that was executed.
Timestamp	The time in which the activity is executed.
Resource	The id of the person that executes an activity.
Amount	The total amount of the fine (can increase due to penalties).
Article	The law/article that was violated.
Expense	The cost of sending the fine to the offender.
Vehicle Class	Car (A), truck (C) or motorbike (M).
Vehicle Type	The brand and model of the car.
Points	The number of points that are deducted from the offender's license.
Payment Amount	The amount of a payment to the local police.
Total Payment Amount	The total amount paid to the local police.
Notification Type	The recipient of the notification: offender (C) or car owner (P).
Delay Send	The time since a fine is created until it is sent to the offender.
Delay Judge	The time since a fine is received by the offender until an appeal to a judge is filed.
Delay Prefecture	The time since a fine is received by the offender until an appeal to a prefecture is filed.
Next Activity	The next task that was executed for that fine.
Duration	The duration of a task.
Elapsed Time	The time since the start of a fine until the task is executed.

Table 2.2: A fragment of the *road fines* event log represented as a table: each row corresponds to an event (shown in the *event id* column) and each column corresponds to an event attribute. Events with the same *fine id* correspond to the same instance of the process. The elapsed time is measured in days.

event id	**fine id**	**activity**	timestamp	amount	next activity	elapsed time	...
...	...	...	...	...	...	...	...
001	A1	create fine	24-07-2006	35	send fine	0	...
002	A1	send fine	05-12-2006	35	-	134	...
003	A100	create fine	02-08-2006	35	send fine	0	...
004	A100	send fine	12-12-2006	35	insert fine notif.	132	...
005	A100	insert fine notif.	15-01-2007	35	add penalty	166	...
006	A100	add penalty	16-03-2007	71.5	send for credit col.	227	...
007	A100	send for credit col.	30-03-2009	71.5	-	972	...
008	A1007	create fine	28-08-2006	21	send fine	0	...
009	A1007	send fine	15-12-2006	21	insert fine notif.	110	...
010	A1007	insert fine notif.	07-01-2007	21	add penalty	133	...
011	A1007	add penalty	08-03-2007	42.5	payment	192	...
012	A1007	payment	30-11-2007	42.5	-	460	...
013	A10082	create fine	11-03-2007	36	payment	0	...
014	A10082	payment	11-03-2007	36	-	0	...
...	...	...	...	...	...	...	...

Table 2.2 shows a fragment of this event log. Note that only a few events and attributes are shown.

The basic flow of a road fine is described as follows: The process starts when a fine is created. After this, the local police has 90 days to send a notification to the offender if the fine is not fully paid. After being notified, offenders have a choice: they can pay the fine, or they can appeal to a judge/prefecture. If they decide to pay, they have 180 days to pay the full amount of the fine, otherwise a penalty is added. If they decide to appeal, they have 60 days to do so. If a fine is not fully paid and an appeal is not filed, the fine is eventually sent to credit collection and the process finishes. Note that the offender can make (partial) payments at any time.

Given the event log shown above in Table 2.2, Figure 2.6 shows a transition system created using the state representation function $r^s(\sigma) = \#_{activity}(\sigma(|\sigma|))$ and the activity representation function $r^a(e) = \#_{activity}(e)$. In other words, both the states and transitions consist of only the last activity that was executed. Note that the transition system was filtered after its creation in order to obtain a simpler representation of the process: states and transitions that occurred in less than 5% of the traces are hidden.

Figure 2.6: Transition system illustrating the road fines management process.

Part II

Foundations

Chapter 3

Process Cubes

Classical process mining techniques focus on analyzing a process through processing its corresponding event log as a whole. Processes inherently contain variability (as discussed in Section 1.2) which can complicate the analysis of such a process. In this thesis, we claim that variability can be dealt with by splitting event data into *process variants* by using event dimensions in such a way that differences (hence, the variability) between variants are exposed e.g., when comparing such variants using process comparison tools.

This chapter formalizes the notion of *process cubes* where the event data is presented, organized and split into cells using different dimensions. These dimensions can be *given* in the event data, or can be *derived* from them, or from the context of the process. For example, process variant detection techniques (see Chapter 6) can be used to find process variants and encode such variant information explicitly in the event data as derived dimensions, which can be used by a process cube to split the data into such variants.

A process cube can use any combination of dimensions (given or derived) to split the data into cells, where each cell in the cube contains a set of events that can be converted into a process variant (i.e., a set of traces) which can be used as an input by any process mining technique such as process comparison (presented in Chapter 5) and process mining workflows (presented in Chapter 4). The interactions between these techniques are illustrated in Figure 3.1.

The notion of a *process cube* is related to the well-known OLAP paradigm providing insights into multidimensional data [81, 92]. An OLAP cube is composed of *facts* and *dimensions*. Facts are numerical measures and are the object

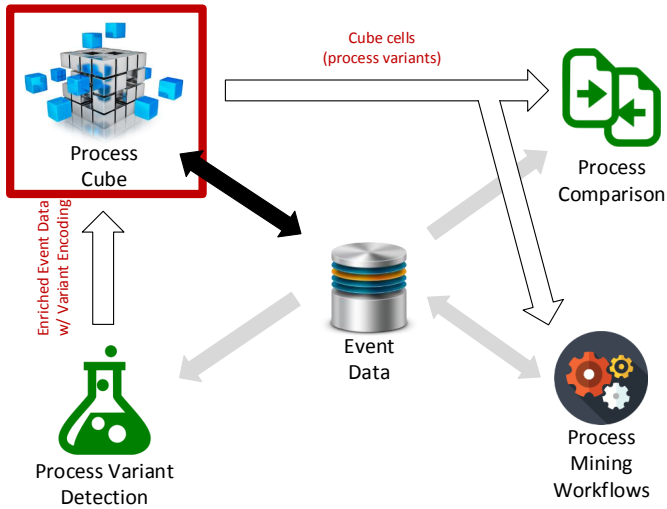


Figure 3.1: Overview of the scope of this chapter: a process cube takes event data as input and splits it into process variants (cells of the cube) that can be used directly by other process mining techniques, such as the ones proposed in this thesis, i.e., process comparison and process mining workflows. Unused interactions are greyed out.

of analysis (e.g., sales, costs, inventory levels). Each fact depends on a set of dimensions, which provide the context for such fact. For example, the dimensions associated with a sale amount (fact) can be the location, product name, and the date when the sale was made. However, dimensions only determine the "structure" of the cube. The combination of dimensions together with *data* can uniquely determine facts. For example, in Table 3.1 fact "1" is fully determined by the dimension values `sale amount` = 10, `city` = "Eindhoven", and `country` = "Netherlands".

Dimensions can be described by a set of attributes related to each other. For example, the *location* dimension may consist of two attributes: `city` and `country`. The attributes in a dimension may be related to each other through a *hierarchy* that defines the aggregation relations of attributes of the dimension. For example, the hierarchy `city` $\rightarrow$ `country` defines that countries can aggre-

gate cities. This defines the *level of detail* with which the facts are presented. In the previous example, sales can be grouped by `city` (more detailed, less aggregated) or by `country` (less detailed, more aggregated).

The main purpose of On-Line Analytical Processing (OLAP) is to provide analytic results over large collections of facts in *real-time*. On the other hand, On-line Transactional Processing (OLTP) query facts on-demand (i.e., for a given analysis) and will most likely not produce results quickly enough. In order to achieve a *real-time* response, most of the processing of the raw facts is done *a priori* (i.e., when the cube is being built) and only once, through aggregation and summarization of these facts over the available dimensions. After an OLAP cube is built, results can be quickly retrieved by using these summarized (pre-calculated) facts. Example 3.2 illustrates the aggregation and summarization of facts in OLAP and how the cube can retrieve results using summarized facts.

Example 3.1 (Data Aggregation and Summarization in OLAP)**.** Let us consider a simple OLAP cube that contains *sales* facts and a *location* dimension that defines where a sale was made.

Each *fact* of this cube is characterized by a measure (i.e., `Sale Amount`), and the *location* where it happened, described by the `City` and `Country` attributes, as shown in the following table:

Table 3.1: Facts in the Cube

Fact Id	Sale Amount	City	Country
1	10	Eindhoven	Netherlands
2	60	Eindhoven	Netherlands
3	53	Amsterdam	Netherlands
4	41	Madrid	Spain
5	32	Sevilla	Spain
6	15	Sevilla	Spain
7	65	Zaragoza	Spain

The *location* dimension is defined by the attribute hierarchy: `City` $\rightarrow$ `Country`, which means that sales associated to cities can be aggregated to countries.

Facts can be directly queried from the cube, as in OLTP (e.g., using SQL). For example, if we want to obtain the total sales amount in *Eindhoven*, we can query the cube and aggregate all the facts that are related

to the city *Eindhoven* (i.e., facts 1 and 2) resulting in a total value of 70. However, in large cubes with many facts and dimensions, this type of ad-hoc querying can become unfeasible in practice.

Before a cube is used, facts can be *pre-aggregated* over dimensions using similar queries, and such pre aggregations can be re-used for many future queries over the cube. For example, if we want to obtain the total sales amount of Spanish cities that are **not** *Sevilla*, instead of calculating the total sales amount of all Spanish cities except *Sevilla* and then adding them, we can use the pre-aggregated value of the total sales amount of the country *Spain* (i.e., 153) , and then substract the pre-aggregated total sales amount of the city *Sevilla* (i.e., 47), resulting in a value of 106.

The pre-aggregation of facts in OLAP is key for its *real-time* response performance. The main advantage of OLAP over OLTP is that the querying is generally done only once for OLAP. Naturally, pre-aggregated values are stored in data structures within the cube. The concrete data representation used for storing pre-aggregations depends on many factors (e.g., the vendor, type of dimensions). In this example, we will use a graph representation to illustrate data aggregation on an OLAP cube.

The graph presented in the following figure shows the pre-aggregation of facts performed over the *location* dimension:

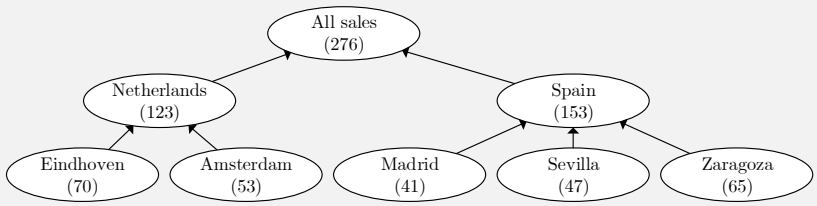


Figure 3.2: Illustration of OLAP aggregation and summarization of facts over a *Location* dimension. Each node contains the aggregated sales of its corresponding city or country. For example, the city *Eindhoven* has a total sales amount of 70 (i.e., facts 1 and 2).

Each node in this graph contains a single *summarized* value that represents the *aggregation* of sale amounts for that location through an *addition* function, and each arc defines an aggregation direction. The bottom row of nodes in the graph describes the aggregated sales for each `City`, which is the lowest level in the hierarchy of the *location* dimension. Note that facts

1 and 2 are related to the city *Eindhoven*. However, on the pre-aggregation graph, only the summarized value 70 (i.e., the sum of the sale amounts of facts 1 and 2) is stored for the city *Eindhoven*. Similarly, the same procedure applies to every value combination of the *location* dimension (i.e., value combinations of the attributes `City` and `Country`). For example, *Spain* has a total amount of sales of 153, which comprehends facts 4, 5, 6 and 7. Let us say that the initial state of the cube is to group all sales together (i.e., the *All sales* node in the pre-aggregation graph). If the user wants to *drill-down* to group facts by `Country`, then the cube does not query the total sales for each country from the raw facts, but uses the summarized values shown in Figure 3.2 to retrieve the answer for each country and/or city.

Research on OLAP is plentiful. An extensive overview of OLAP approaches is presented in [35]. The application of OLAP on non-numerical data is increasingly being explored in recent years. Temporal series, graphs, and complex event sequences are possible applications [36, 105, 107].

Process cubes can be seen as an adaptation of OLAP to the use of *events* instead of *facts* [155]. The difference between a fact being a numerical measurement and an event being a more complex type of data causes two important differences between OLAP and process cubes: *Aggregability* and *Summarizability*.

Aggregability refers to the fact that, in OLAP, facts are numerical measures that can be *aggregated* through numeric algebraic functions, e.g., sum, average. Events cannot be aggregated using numeric algebraic functions. However, events can be (pre-)aggregated and grouped using set operations (e.g., union), where the result of an aggregation of events is simply a set of events. Aggregation in OLAP (and process cubes) is not always possible [92]. The dimensions used in the cube usually have to satisfy some properties such as completeness (i.e., a fact or event has a value for all the attributes of the dimension) and disjointness of attributes (e.g., in a *location* dimension, a *city* cannot be related to different *countries*).

Summarizability in OLAP refers to the aggregation of facts for reducing a set of (aggregated) values into a single *summarized* value e.g., mode, average of values. Some authors have studied summarizability issues in OLAP [117, 120] where facts cannot always be summarized, and attempt to solve it by introducing rules and constraints to the data model. However, process cubes have to deal with a much more complex representation of data. Many events cannot be reduced into a single event.

In summary, a process cube handles events like an OLAP cube would handle

non-aggregatable facts. To illustrate this, let's go back to Example 3.2. If each fact in Table 3.1 is considered as an event (i.e., non-summarizable data point), the pre-aggregation graph shown in Figure 3.2 should contain, for each node, the set of facts that are related to it, instead of a summarized value. For example, the node *Eindhoven* should only contain the facts 1 and 2 regardless of the value of such facts.

Events in a process cube can be aggregated using *set operations* (e.g., set addition) where the elements of the sets are events. For example, sales in the city *Eindhoven* (i.e., facts 1 and 2 in Table 3.1) and Amsterdam (i.e., fact 3 in Table 3.1) can be aggregated into the sales in the country *Netherlands* (i.e., fact 1, 2 and 3 in Table 3.1). Note that this only helps to identify which events should be considered for any query.

Naturally, this type of non-summarized pre-aggregation has an impact on the performance and *real-time* responsiveness of an OLAP cube: non-summarized pre-aggregation will likely have a slower response time than summarized pre-aggregation. However, in process cubes this presents a dramatic advantage over not pre-aggregating events: after a pre-aggregated process cube is built, cube operations do not require to retrieve events from storage in order to evaluate and distribute them into the cells of the cube, as the relevant events for each cell can be identified using simple set operations over pre-calculated sets of events.

The remainder of this chapter is structured as follows. Section 3.1 discusses related approaches. Section 3.2 formally defines a process cube, its operations and how the cells of the cube can be translated into event logs. Section 3.3 describes the implementation of the approach. Section 3.4 shows the application of process cubes to the running example 2.4 and shows the interaction of the process cube tool with other process mining techniques proposed in this thesis. Note that the full application of process cubes to real case studies and the insights that can be retrieved is presented in depth in Part IV. Finally, Section 3.5 concludes the chapter.

3.1 Related Work

The idea of applying OLAP techniques to event data has been recently approached by some authors.

The *event cube* approach described in [128] presents an exploratory view on the applications of OLAP operations using *events*. The first description of a *process cube* was introduced in [155]. Later, a data-warehouse-based process cube approach was presented in [184]. The process cube notion has been proven

useful in several case studies [5, 161, 183]. These approaches have established a *conceptual* framework for process cubes, however, they still present some conceptual limitations, discussed as follows.

The work presented in [155] has two main limitations. The first limitation is related to the fact that derived attributes are created directly on the *event base* (instead of on the cube structure) which may be used with many *process cube structures*. This would force all the dimensions that correspond to a specific event attribute to have exactly the same meaning and value set. For example, it is not possible to create a derived attribute *customer type* in different *process cube structures* according to different criteria: in one process cube structure the VIP customers must have an income over 1000 and in the other cube structure the income must be over 2000. This is because the derived attribute *customer type* can be calculated only once and is added as an extra event attribute in the event base. The second limitation is the lack of attributes within the dimensions i.e., there are no attribute hierarchies or compositions, which are necessary for aggregation. Also, there are no defined granularity levels within the dimensions, which are necessary to perform cube operations such as *roll-up* and *drill-down*. Note that the approach presented in [155] allow for events to be present in multiple cells, which is not allowed in our approach in order to enable the aggregability of cells.

The work presented in [184] (extended afterwards in [182] with improved performance and interactivity) is tightly coupled to a data warehouse implementing a snowflake schema, where the *trace identifier* and the *activity identifiers* are fixed and are central to all dimensions. This means that if the analyst wants to change the activity identifier to e.g., analyze social networks (where the *resource* is the activity identifier), the whole process cube becomes useless and has to be rebuilt again, which can take significant time and resources. Also, traces (instead of events) are the basic elements in the cube. This means that traces cannot be *horizontally split* into e.g., front-office and back-office parts of a process.

In the remainder of this chapter, an improved formalization of the *process cube* concept is presented, which addresses the limitations discussed above.

3.2 Process Cubes

A process cube is composed of two main components: A *process cube structure* and a compatible *event base*. The process cube structure defines the *dimensions* of the cube (i.e., the schema) and the event base contains the collection of

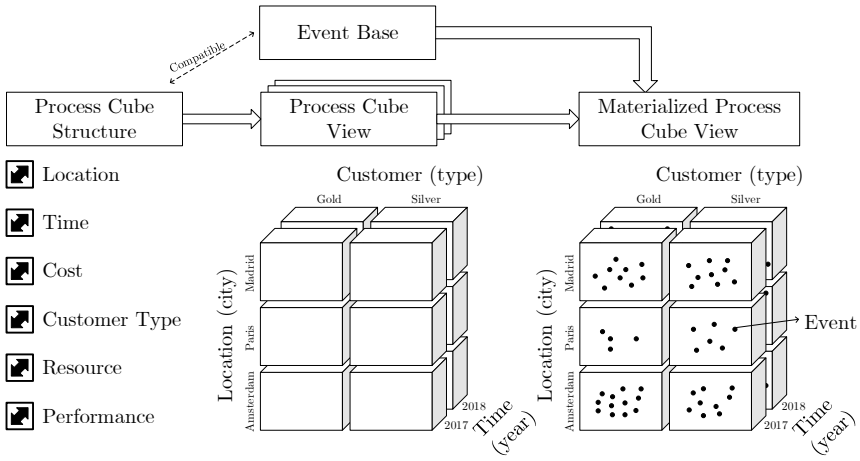


Figure 3.3: Overview of a process cube and its components.

events (i.e., the data) that will be used to fill the cube.

Figure 3.3 shows an overview of a process cube and its components. Once a *process cube structure* is defined, several *process cube views* can be obtained from it using *process cube operations*, e.g., *slice* and *dice* (see Section 3.2.4). A process cube view defines the visible cells of the cube. Using a compatible *event base*, the cells of the cube are filled with their matching events to become a *materialized process cube view*.

The remainder of this section is organized as follows. Section 3.2.1 describes the structure of a process cube and the different views that can be obtained from it. Section 3.2.2 describes the event base (i.e., the data source for the cube) and the compatibility between an event base and a process cube structure. Section 3.2.3 describes how the cells in the process cube view are filled with event data and how the cells can be transformed into event logs. Finally, Section 3.2.4 discusses how traditional OLAP operations such as *slice* and *dice* can be applied in process cubes.

3.2.1 Process Cube Structure

The *structure* of a process cube is independent of the actual data. A process cube structure is fully characterized by a set of *dimensions*. Before formally defining

what a dimension is, it is necessary to introduce the concept of directed acyclic graphs.

Definition 3.1 (Directed Acyclic Graph)**.** *A directed acyclic graph (DAG) is a pair* (N, A) *where* N *is a set of nodes and* $A \subseteq N \times N$ *is a set of arcs connecting these nodes, where:*

- *The arcs are* directed*:* $(n_1, n_2) \in A$ *is a* directed *arc that starts in* n_1 *(i.e., the* source *node) and ends in* n_2 *(i.e., the* target *node).*
- *The graph has a* topological order*: A topological order of a directed graph* (N, A) *is a sequence* $\sigma \in N^*$ *so that for every directed arc* $(\sigma_i, \sigma_j) \in A \Rightarrow i < j$

Note that multiple topological orders can exist for the same DAG e.g., in the presence of partial orders. Therefore, a DAG can often be mapped onto multiple sequences. Also note that the graphs must be acyclic to be used as a Dimension in order to have finite-length paths within it. Now, we can define dimensions.

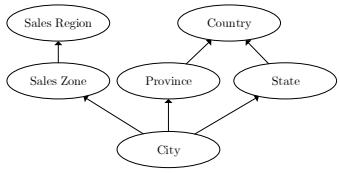
Definition 3.2 (Dimension)**.** *Let* $\mathcal{N}$ *be the universe of attribute names and* $\mathcal{V}$ *the universe of values. A* dimension $d = ((A, H), valueset)$ *consists of a* hierarchy (A, H) *which is a directed acyclic graph where* $A \subseteq \mathcal{N}$ *correspond to dimension attributes and* $H \subseteq A \times A$ *corresponds to a set of directed edges (i.e., hierarchical relations), and a function* $valueset : A \to \mathbb{P}(\mathcal{V})$ *defining the possible set of values for each attribute. The universe of all possible dimensions is denoted as:*
$\mathcal{D} \subseteq \big(\mathbb{P}(\mathcal{N}) \times \mathbb{P}(\mathcal{N} \times \mathcal{N})\big) \times (\mathcal{N} \nrightarrow \mathbb{P}(\mathcal{V}))$

The attributes in A in a dimension $((A, H), valueset)$ are unique. Note that for a dimension d, we denote its set of attributes as A_d. The set of directed edges H defines the hierarchy relations between the attributes of the dimension. An edge $(a_1, a_2) \in H$ means that attribute a_1 can be *rolled up* (i.e., aggregated) to attribute a_2 (see Section 3.2.4 for a detailed description).

A dimension should describe events from a single perspective through any combination of its attributes (e.g., attributes `city` and `country` can describe a *Location*) where attributes describe the perspective from higher or lower levels of detail (e.g., `city` describes a *Location* in a more fine-grained level than `country`). However, this is not strict and users can define dimensions as they want.

A dimension attribute $a \in A$ has a $valueset(a)$ that is the set of possible values and typically only a subset of those values are present in a concrete instance

of the process cube. Note that $valueset(a)$ can define an enumeration of elements (e.g., $valueset(customerType) = \{vip, regular\}$ for $a = customerType$), but also can define ranges e.g., $valueset(age) = \{i \in \mathbb{N} | 0 \leq i \leq 120\}$ for $a = age$, or $valueset(temperature) = \{t \in \mathbb{R} | -40 \leq t \leq +60\}$ for $a = temperature$. Another example: $valueset(cost) = \mathbb{N}$ allows for infinitely many possible values. It is important to note that the value sets of the different attributes of a dimension are not typed or predefined. For example, in a *Location* dimension, it is possible to have a value $Eindhoven \in valueset(City)$ with $Netherlands \notin valuset(Country)$. The relations between the attribute values are not defined here, but are taken explicitly from the event data that is used to "materialize" the cube. Figure 3.4 shows an example of a *Location* dimension. Figure 3.5 shows an example of an *Organization* dimension.

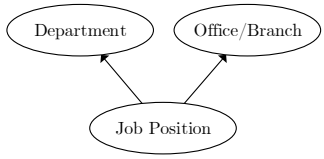


(a) Attribute Hierarchy

Attribute	Value set
Sales Region	{North Europe, Latin America, etc.}
Sales Zone	{Scandinavia, Benelux, etc.}
Country	{Netherlands, Chile, Singapore, USA, etc.}
Province	{Noord Brabant, Andalucía , Flanders, etc.}
State	{Washington, California, etc.}
City	{Talca, Paris, London, etc.}

(b) Attribute Value sets

Figure 3.4: Example of a *Location* dimension.



(a) Attribute Hierarchy

Attribute	Value set
Office	{Boston, London, etc.}
Department	{Marketing, Operations, National Sales, etc.}
Job Position	{Programmer, Sales Executive, etc.}

(b) Attribute Value sets

Figure 3.5: Example of an *Organization* dimension.

Definition 3.3 (Process Cube Structure)**.** *Let $\mathcal{D}$ be the universe of dimensions. A process cube structure is a set of dimensions $PCS \subseteq \mathcal{D}$, where for any two dimensions $d_1 = ((A_1, H_1), valueset_1) \in PCS$ and $d_2 = ((A_2, H_2), valueset_2) \in PCS : d_1 = d_2 \vee A_1 \cap A_2 = \emptyset$.*

All dimensions in a process cube structure are independent from each other. This means that they do not have any attributes in common. However, the value sets of the attributes might have common values. We introduce the following notation to refer to the union of all the sets of dimension attributes of the dimensions contained in the process cube structure PCS:

$$A_{PCS} = \bigcup_{d \in PCS} A_d$$

where A_d is the set of the attribute names in the dimension d.

Once a process cube structure is defined, it does not change. While applying typical OLAP operations such as slice, dice, roll up and drill down (defined in Sec 3.2.4) we only change the way we are visualizing the cube and its content.

A process cube structure can contain many dimensions. A dimension can contain many attributes, and each dimension attribute can be related to many different attribute values. However, not all dimensions, attributes and values are always relevant. Often, the analyst wants to focus only on some dimensions, and on specific parts of them. For example, the analyst might want to focus only on the *location* dimension (shown in Figure 3.4), and might want to focus on analyzing different cities. In such case, only the *location* dimension is *visible* i.e., it will be used to define the visible part of the cube (see Definition 3.6). Given a process cube structure PCS, we define the subset of *visible* dimensions to be visualized in the cube as $D_{vis} \subseteq PCS$. Although the term *cube* suggests a three dimensional object, a process cube can have any number of visible dimensions.

Given a set of visible dimensions D_{vis}, for every visible dimension $d = ((A_d, H_d), valueset_d) \in D_{vis}$, we must choose the attribute $a \in A_d$ that defines the *granularity* (i.e., level of detail) of the dimension. For example, in the *location* dimension shown in Figure 3.4, the highest level of granularity corresponds to the `city` attribute (i.e., the most detailed) and the lowest level of granularity corresponds to both `country` and `sales region` (i.e., the least detailed). Note that the highest or lowest granularities of a dimension are determined by the attribute hierarchy H (see Figure 3.4.a).

Definition 3.4 (Granularity of Visible Dimensions)**.** *Given a process cube structure PCS and a set of visible dimensions $D_{vis} \subseteq PCS$, the granularity of the visible dimensions is defined through a function that maps visible dimensions to attributes of such dimensions, and is defined as: $gran : D_{vis} \rightarrow A_{PCS}$, such that for any $d \in D_{vis} : gran(d) \in A_d$.*

These granularities are also used for defining the visible part of the cube (see Definition 3.6). For example, the *location* dimension shown in Figure 3.4 can

be aggregated by `country`, `province`, `city`, etc. For example, if the `country` is selected as the granularity of the *location* dimension, then the cells in the cube will relate to different countries. Alternatively, if the `city` is selected as the granularity, then the cells in the cube will relate to different cities.

Let us say that now the analyst wants to only focus on cities within the province of Noord Brabant. This means that, within the *location* dimension (shown in Figure 3.4), the only value that should be considered for the attribute `province` is "Noord Brabant". This means that only events that happened in the province of Noord Brabant should be selected.

Definition 3.5 (Selection of Dimension Attribute Values)**.** *Given a process cube structure PCS, the selection of dimension attribute values is performed using the function:*
$sel : A_{PCS} \rightarrow \mathbb{P}\big(\bigcup_{a \in A_{PCS}} valueset(a)\big)$ *such that for any dimension* $d \in PCS$: $a \in A_d \Rightarrow sel(a) \subseteq valueset(a)$.

Note that only values that exist in the $valueset$ of a dimension attribute can be selected. Also note that the sel function can be applied to any attribute of any dimension in the cube structure, regardless of whether the dimension is visible or not. The sel function does not depend on the visible dimensions or the granularities defined for them: it can be used to select values of any attribute in any dimension of the cube. For example, in the dimension *location* in Figure 3.4, one could select the `province` of Noord Brabant, and in a invisible *time* dimension, one could select the `year = 2018` to select only the events that occurred in 2018 in the province of Noord Brabant. One can even select values for different attributes of the same dimension. In our approach, we made this selection as flexible as possible, so it is up to the user to check if the selection is done properly. Note that if this selection is done incorrectly, it might lead to empty results. For example, in the dimension *Location* in Figure 3.4, one could select the `city = Eindhoven` and the `country = Spain` and this would produce empty results since no event can have both values.

These selections, in combination with the granularity levels discussed above, can define the visible part of the process cube structure. For example, if we have a cube with $D_{vis} = \{$*location*$\}$, where $gran(location) =$ `city` and $sel(province) = \{$*Noord Brabant*$\}$, the cells of the resulting process cube view will correspond to different cities within the province of Noord Brabant.

A *process cube view* defines the visible part of the process cube structure.

Definition 3.6 (Process Cube View)**.** *Let PCS be a process cube structure, $D_{vis} \subseteq PCS$ be the set of visible dimensions, sel be a selection function (see Definition 3.5)*

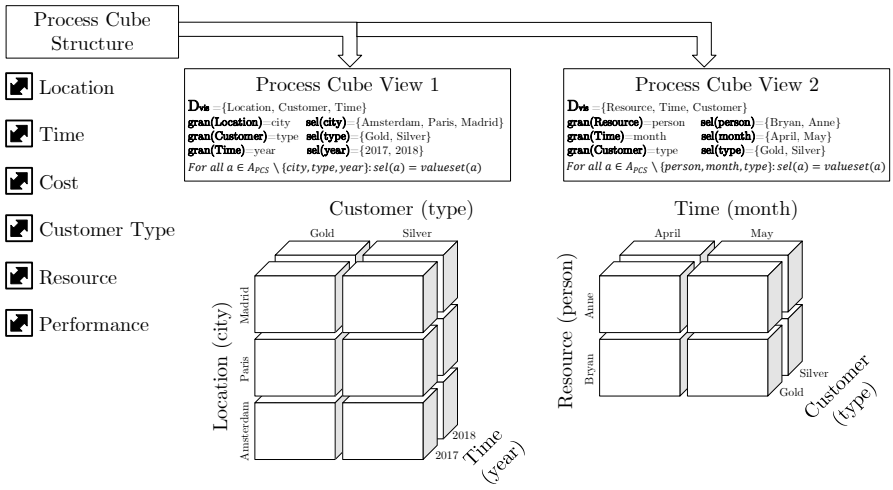


Figure 3.6: Example of different *process cube views* (PCV) obtained from the same process cube structure.

and $gran$ be a granularity function (see Definition 3.4). A process cube view is simply a triplet $PCV = (D_{vis}, sel, gran)$.

Many different process cube views can be obtained from the same process cube structure. For example, Figure 3.6 shows two process cube views obtained from the same process cube structure. The cells of the process cube view 1 are defined by the *Location*, *Time*, and *Customer* dimensions. The cells of the process cube view 2 are defined by the *Resource*, *Customer*, and *Time* dimensions. Note that the granularity level of the *Time* dimension in the process cube view 1 is set to `year`, and in the process cube view 2 is set to `month`.

Definition 3.7 (Cell Set)**.** *Let PCS be a process cube structure and $PCV = (D_{vis}, sel, gran)$ be a view over PCS. CS_{PCV} is the* cell set *of PCV, and is defined as the set of mappings:*

$$CS_{PCV} = \left\{ f : \{gran(d) | d \in D_{vis}\} \to \mathcal{V} \middle| \forall_{d \in D_{vis}} : f\big(gran(d)\big) \in sel\big(gran(d)\big) \right\}$$

Each mapping in the cell set CS defines a visible *cell* of the process cube view. For example, the process cube view 1 shown in Figure 3.6 with visible dimensions *Location*, *Customer* and *Time* with their granularity set to: *gran(Location)=*

`City`, *gran(Customer)*= `Type` and *gran(Time)*= `Year` and the selected values of those attributes are set to: *sel(City)* = {*Amsterdam, Paris, Madrid*}, *sel(Type)* = {*Gold, Silver*} and *sel(Year)* = {*2017,2018*} has a cell set with the following 12 mappings (i.e., cells):

$$\Big\{\{(\textit{City, Amsterdam}), (\textit{Year, 2017}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Amsterdam}), (\textit{Year, 2017}), (\textit{Type, Silver})\},$$
$$\{(\textit{City, Amsterdam}), (\textit{Year, 2018}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Amsterdam}), (\textit{Year, 2018}), (\textit{Type, Silver})\},$$
$$\{(\textit{City, Paris}), (\textit{Year, 2017}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Paris}), (\textit{Year, 2017}), (\textit{Type, Silver})\},$$
$$\{(\textit{City, Paris}), (\textit{Year, 2018}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Paris}), (\textit{Year, 2018}), (\textit{Type, Silver})\},$$
$$\{(\textit{City, Madrid}), (\textit{Year, 2017}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Madrid}), (\textit{Year, 2017}), (\textit{Type, Silver})\},$$
$$\{(\textit{City, Madrid}), (\textit{Year, 2018}), (\textit{Type, Gold})\},$$
$$\{(\textit{City, Madrid}), (\textit{Year, 2018}), (\textit{Type, Silver})\}\Big\}$$

3.2.2 Event Base as a Data Source for the Cube

The starting point for most process mining techniques is an *event log*. These logs are created having a particular process and a set of questions in mind. The difference between an event log and a simple set of events is that in the first, events are organized in traces (using an attribute as *trace id*) and are related to an activity or class (using an attribute as *activity id*). The relations and ordering between events are subject to the attributes that are used as *trace id* and *activity id*. For example, events that are in the same trace under a given *trace id* may not be in the same trace if another attribute is used as *trace id*. An *event collection* is a set of events that have certain attributes, but no defined notion of *traces* and *activities*.

An *event base* is a large collection of events not tailored towards a particular process or predefined set of questions. An event base can be seen as an all-encompassing event log or the union of a collection of related event logs. The events in the event base are used to populate the cube.

Definition 3.8 (Event Base)**.** *An event base is a triplet* $EB = (E, P, \#)$ *where* $E \subseteq \mathcal{E}$ *is a set of events,* $P \subseteq \mathcal{N}$ *is a set of event attributes, and* $\# : P \to (\mathcal{E} \nrightarrow \mathcal{V})$ *is a function that relates attributes to values for given events (see Definition 2.9).*

Any event log L with a set of attributes P and a function $\#$ as described above, can be trivially converted to the event base $EB_L = (E_L, P, \#)$ where $E_L = \bigcup_{\sigma \in L} \{e \in \sigma\}$.

A process cube structure PCS and an event base EB are independent elements, where the PCS is the structure and the EB is the content of the cube. To make sure that they can be used together, they need to be related through a mapping function and then we check whether they are compatible.

Definition 3.9 (Mapper)**.** *A mapper is a triplet $M = (PCS, EB, R)$ where PCS is a process cube structure, $EB = (E, P, \#)$ is an event base and R is a function defined as $R : A_{PCS} \rightarrow (\mathbb{P}(P) \times (E \nrightarrow \mathcal{V}))$.*

For any dimension attribute $a \in A_{PCS}$, $R(a)$ is defined as the pair (P_a, g_a) where $g_a : E \nrightarrow \mathcal{V}$ is a dimension attribute value calculation function that, for a given dimension attribute a, maps events to values by using the values of a defined set of event attributes $P_a \subseteq P$, such that:

1. *g_a only depends on the attributes in P_a:*

$$\forall_{e_1, e_2 \in E} : \Big(\big(\forall_{p \in P_a} : \#_p(e_1) = \#_p(e_2) \big) \Rightarrow g_a(e_1) = g_a(e_2) \Big)$$

2. *g_a is undefined for an event if one of the considered attributes of that event is undefined:*

$$\forall_{e \in E} : \Big(\exists_{p \in P_a} : \big(\#_p(e) = \bot \big) \Rightarrow \big(g_a(e) = \bot \big) \Big)$$

Note that each dimension attribute is related to one set of event attributes which is used to calculate the value of the dimension attribute for any event in the event base through a specific calculation function. For example, in a *Customer* dimension, we can calculate an *age* dimension attribute using the event attributes *time* and *birthday* by defining:

$$R(age) = \big(\{time, birthday\}, g_{age} \big)$$

such that $g_{age} : E \nrightarrow \mathcal{V}$ with $g_{age}(e) = \#_{time}(e) - \#_{birthday}(e)$ for any $e \in E$. As another example, in a *Time* dimension, a dimension attribute `day type` (e.g., weekday, weekend) can be calculated using the event attributes `{day,month,year}` according to some specific calendar rules.

A set of event attributes can be used by more than one dimension attribute producing different results if the calculation function is different. For example,

in sales one could use the set of event attributes {*purchase amount, purchase num*} to classify customers into a dimension attribute *customer type* = {*Gold, Silver*} (i.e., if *purchase amount* > 50 and *purchase num* >10, then *customer type* = *Silver*) and at the same time to detect fraud into a dimension attribute *fraud risk* = {*High,Low*} (i.e., if *purchase amount* > 100000 and *purchase num* $= 1$, then *fraud risk* = *High*).

Given a mapper $M = (PCS, EB, R)$ we say that PCS and EB are *compatible* through R, making all the possible views of PCS also compatible with the EB.

3.2.3 Materializing a Process Cube View

Once we selected a part of the cube structure through a process cube view, and there is a cell set defined as the visible part of the cube (see Definition 3.7), now we have to add content to those cells. In other words, we have to add *events* to these cells so they can be used by process mining algorithms.

Definition 3.10 (Materialized Process Cube View)**.** *Let $M = (PCS, EB, R)$ be a mapper with PCS being a process cube structure, $EB = (E, P, \pi)$ being an event base, and R being a mapping function such that for any $a \in A_{PCS}, R(a) = (P_a, g_a)$. Let $PCV = (D_{vis}, sel, gran)$ be a view over PCS with a cell set CS_{PCV}. The* materialized process cube view *for PCV and EB is a defined as a function $MPCV_{PCV,EB} : CS_{PCV} \rightarrow \mathbb{P}(E)$ that relates mappings in the cell set (i.e., cells) to sets of events, so that $\forall_{c \in CS_{PCV}}$: $MPCV_{PCV,EB}(c) =$*

$$\Big\{ e \in E | \big(\forall_{a \in A_{PCS}} : g_a(e) \in sel(a)\big) \wedge \big(\forall_{d \in D_{vis}} : c(gran(d)) = g_{gran(d)}(e)\big) \Big\}$$

Figure 3.3 shows an example of a materialized process cube view. Each of the selected dimensions conform the cell distribution of the cube, and the events in the event base are mapped to these cells.

For example, for a mapping (i.e., cell) c = {(*year,2017*),(*city,Amsterdam*)} one could relate all events in the event base that have both attribute values to that cell, as long as their attribute values are part of the sel function defined by the process cube view.

Note that $MPCV_{PCV,EB}(c)$ only yields a set of events as a result. However, most process mining techniques rely on event logs where events are grouped in traces. Therefore, we need to be able to transform this set of events so that for any given cell in the cube, we can obtain an event log from it.

In order to do this, the first step is to group events by their "case id" attribute. Given any mapping $c \in CS_{PCV}$, a related set of events $MPCV_{PCV,EB}(c)$ and

an attribute $a \in A_{PCS}$ selected as "case id", for any "case id" value $v \in sel(a)$ we denote the set of events that have such an attribute value as:

$$E_v = \{e \in MPCV_{PCV,EB}(c) | \#_a(e) = v\}$$

Each of these sets contains all the events in the cell that have the same value for a given "case id" attribute.

Now, we have to transform sets of events into traces. Given a pair $(E, \prec)$, where E is a set of events and $\prec$ is a total order on events, we define the function $seq : \mathbb{P}(E) \rightarrow E^*$ that maps sets of events to sequences, defined as:

$$seq(E) = \sigma \text{ such that } E = \{e \in \sigma\} \wedge \forall_{1 \leq i < j \leq |\sigma|} : \sigma(i) \prec \sigma(j)$$

Note that σ contains all the events in E. By using the elements described above, we can transform a set of events related to a cell into an event log. Given any mapping $c \in CS_{PCV}$, a related set of events $MPCV_{PCV,EB}(c)$ and an attribute $a \in A_{PCS}$ selected as "case id", c can be mapped to the event log L_c defined as:

$$L_c = \bigcup_{v \in sel(a)} \{seq(E_v)\}$$

Note that this approach allows for any attribute $a \in A_{PCS}$ to be used as the "case id" for grouping events into traces. For example, in hospital setting, a = "Patient ID" can be used to group events by patient, while a = "Doctor ID" can be used to group events by the treating doctor. This allows for analyzing event data from different points of view.

Given the above definitions, we can specify the *operations* that can be done over a process cube view in order to perform multidimensional exploration of event data.

3.2.4 Process Cube Operations

In this section, we adapted the classical OLAP operations to the context of process cubes (i.e., to work with events instead of numerical facts). The *slice operation* produces a new cube by allowing the analyst to filter (pick) specific values for attributes within a cube dimension d, while removing d from the visible part of the cube.

The *dice operation* produces a sub-cube by allowing the analyst to filter (pick) specific values for one of the dimensions. No dimensions are removed in this case, but only the selected values are considered. Figure 3.7 illustrates the

notions of slicing and dicing. For both operations, the same filtering is applied. In the case of the slice operation, the *Time* dimension is no longer visible, but after a dice operation one could still use that dimension for further operations (e.g., drilling down to *month*) keeping the same dimensions visible.

The *roll up* and *drill down* operations do not remove any dimensions or filter any values, but only change the level of granularity of a specific dimension. Figure 3.8 illustrates the concept of drilling down and rolling up. These operations are intended to show the same data with more or less detail (granularity). However, this is not guaranteed as it depends on the dimension definition.

Definition 3.11 (Slice)**.** *Let PCS be a process cube structure and let $PCV = (D_{vis}, sel, gran)$ be a view of PCS. We define* slice *for a dimension $d = ((A_d, H_d), valueset_d) \in D_{vis}$ and a* filtering *function $fil_d : A_d \to \mathbb{P}(\mathcal{V})$ where for any $a \in A_d$, $fil_d(a) \subseteq valueset_d(a)$, as: $slice_{d,fil_d}(PCV) = (D'_{vis}, sel', gran')$, where:*

- *$D'_{vis} = D_{vis} \setminus \{d\}$ is the new set of visible dimensions, and*
- *$sel' : A_{PCS} \to \mathbb{P}(\mathcal{V})$ is the new selection function, where:*

  for any $a \in A_d, sel'(a) = fil_d(a)$, and

  for any $a \in A_{PCS} \setminus A_d, sel'(a) = sel(a)$.
- *$gran' : D'_{vis} \to \bigcup_{d \in D'_{vis}} A_d$ is the new granularity function, where:*

  for any $d' \in D'_{vis}, gran'(d') = gran(d')$

The *slice* operation produces a new process cube view $(D'_{vis}, sel', gran')$. Note that in the new process cube view, d is no longer a visible dimension, i.e., $d \notin D'_{vis}$, but it can be used for filtering. The new sel' function will still be valid as a value set selection function for filtering even when the corresponding dimension is no longer visible. For any attribute $a \in A_d$, the new selection function takes the values given by the filtering function. For any attribute $a \in A_{PCS} \setminus A_d$ the new selection function is not reset to the valueset of a but keeps the existing selected values (e.g., filtered values from previous slices). This allows to filter values using many attributes simultaneously (even from the same dimension) regardless of the dimensions being sliced. Also, note that the new $gran'$ function yields the same values as $gran$ for all the dimensions in D'_{vis}, but is undefined for d i.e., $d \notin dom(gran')$.

For example, for sales data one could slice the cube for a dimension *Location* for City *Eindhoven*, the Location dimension is removed from the cube and only sales of the stores in *Eindhoven* are considered. One could also do more complex

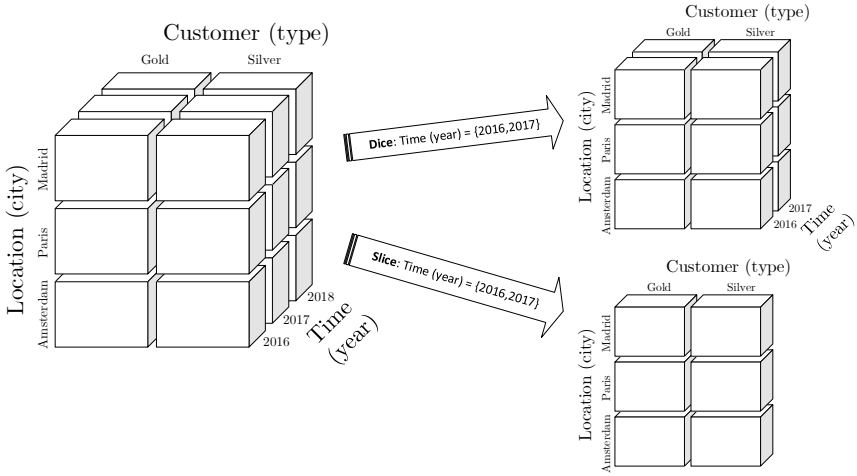


Figure 3.7: Example of a process cube view being similarly sliced and diced, resulting in different process cube views.

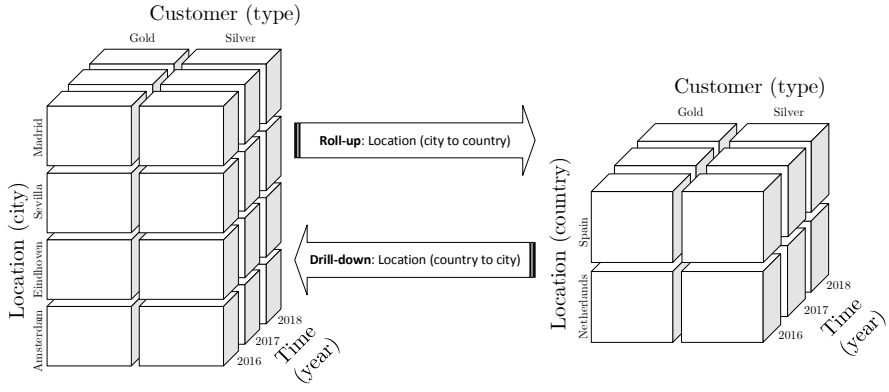


Figure 3.8: Example of process cube view being rolled up and drilled down, changing the level of granularity of the process cube view.

slicing. For example, for a dimension *Time*, one could slice that dimension and select years 2013 and 2014 and months *January* and *February*, then the time dimension is removed from the cube and only sales in *January* or *February* of years 2013 or 2014 are considered.

Definition 3.12 (Dice). *Let PCS be a process cube structure and let $PCV = (D_{vis}, sel, gran)$ be a view of PCS. We define* dice *for a dimension $d = ((A_d, H_d), valueset_d) \in D_{vis}$ and a* filtering *function $fil_d : A_d \rightarrow \mathbb{P}(\mathcal{V})$ where for any $a \in A_d, fil_d(a) \subseteq valueset_d(a)$, as: $dice_{d,fil_d}(PCV) = (D_{vis}, sel', gran)$, where:*

- *$sel' : A_{PCS} \rightarrow \mathbb{P}(\mathcal{V})$ is the new selection function, where:*

  for any $a \in A_d, sel'(a) = fil_d(a)$, and

  for any $a \in A_{PCS} \setminus A_d, sel'(a) = sel(a)$.

The $dice$ operation is very similar to the $slice$ operation defined previously, where the only difference is that in $dice$ the dimension is not removed from D_{vis}, hence the $gran$ function remains unaffected.

The visible dimensions and their granularities define the visible cells in a process cube. We need to be able to change such granularities in order to make new process cube views.

Definition 3.13 (Change Granularity). *Let PCS be a process cube structure and $PCV = (D_{vis}, sel, gran)$ a view of PCS. We define a change of granularity (*chgr*) for a dimension $d = ((A_d, H_d), valueset_d) \in D_{vis}$ and an attribute $a \in A_d$ as: $chgr_{d,a}(PCV) = (D_{vis}, sel, gran')$, where $gran'(d) = a$, and for any $d' \in D_{vis} \setminus \{d\}, gran'(d') = gran(d')$.*

This operation produces a new process cube view and allows us to set any attribute of the dimension d as the new granularity for that dimension, leaving any other dimension untouched. Note that D_{vis} and *sel* always remain unaffected when changing granularity. Typical OLAP cubes allow the user to "navigate" through the cube using roll up and drill down operations, changing the granularity in a guided way through the hierarchy of the dimension. The hierarchy of a dimension defines the granularity relations between its attributes. For example, in Figure 3.8 a *Location* dimension (see Figure 3.4) in a cube is rolled up from a *city* attribute to a *country* attribute. This allows us to view events from a more coarse-grained viewpoint (i.e., split events by *country* instead of by *city*).

Now we define the roll up and drill down operation using the previously defined *chgr* function.

Definition 3.14 (Roll up & Drill down). *Let PCS be a process cube structure, $PCV = (D_{vis}, sel, gran)$ a view of PCS and $d = ((A_d, H_d), valueset_d) \in D_{vis}$ a visible dimension in PCV. We can* roll up *the dimension d if $\exists_{a \in A_d}, (gran(d), a) \in H$. The result is a more coarse-grained cube: $rollup_{d,a}(PCV) = chgr_{d,a}(PCV)$. We can* drill down *the dimension d if $\exists_{a \in A_d}, (a, gran(d)) \in H$. The result is a more fine-grained cube: $drilldown_{d,a}(PCV) = chgr_{d,a}(PCV)$.*

If there is more than one attribute a that the dimension could be *rolled up* or *drilled down* to, then any of those attributes can be a valid target, but we can pick only one each time. For example, in the dimension $Location$ described in Fig 3.4, we could roll up the dimension from *City* to *Province, State* or *Sales Zone*.

3.3 Implementation

The approach presented in this chapter has been implemented as a stand-alone tool called *Process Mining Cube* (**PMC**) and it is freely available at `https://abolt.github.io/ProcessMiningCube/`. The tool provides support for designing and storing a process cube structure and its dimensions through a wizard. A process cube structure can be saved for later use, and the same process cube structure can be used with many event bases, as long as they are compatible.

PMC allows the user to distribute events into the cells of the cube, defined by the selected dimensions and value sets. Each cell is represented by a so-called *metric* (e.g., number of events in the cell) and the events in a cell can be converted into an event log. Such event logs can be visualized using standard state-of-the-art process discovery and log visualization techniques, as shown in Figure 3.9. Alternatively, these event logs can be compared using process comparison techniques, and also can be used as input for process mining workflows.

Given the large collections of events that are available nowadays, PMC uses an embedded SQLite database to store the event base (see Section 3.2.2) on disk.[1] In this internal database, we use dynamic SQL indexing to speed-up the retrieval of events based on attribute values.

All of the interactive and visual components of PMC (e.g., the cell visualizer, wizards) were built using JavaFX8, hence Java 8 is required to run the tool.[2]

[1] SQLite website: `https://sqlite.org/`

[2] JavaFX8 can be found here: `http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html`

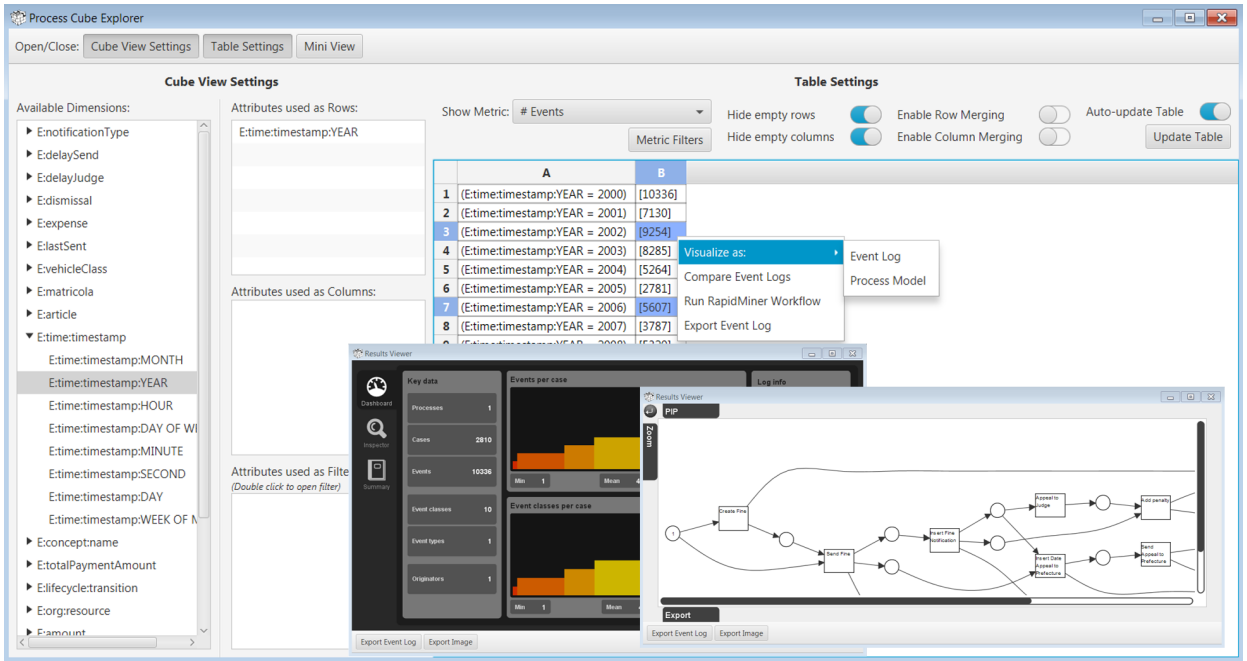


Figure 3.9: Screenshot of the Process Mining Cube (PMC) in action. The cells of the cube show different metrics and can be visualized as process models or event logs. They can be also compared, can be used as input for scientific workflows, and can be used for conformance checking.

The cells of a process cube contain sets of events that can be trivially transformed into an event log. Such event logs can be utilized by existing tools and techniques. PMC has been integrated with two external tools: `ProM` and `RapidProM`.

The integration with ProM provides a wide variety of state-of-the-art process mining techniques. PMC launches a lightweight (i.e., without user interface) instance of ProM in the background, which handles the execution of process mining plugins.

The integration with the analytic workflows tool `RapidMiner` (see Chapter 4) through the RapidProM extension allows PMC to use the contents of any cell(s) of the cube as input to run analytic workflows (with or without process mining components). Therefore, any process mining technique that is incorporated in RapidProM can also be used by PMC. Similarly to the ProM integration, PMC launches a UI-less lightweight instance of RapidProM in the background. PMC allows to select which workflow should be executed using the event data, but does not support the design of such workflow. The design of the workflow should be done a-priori using the `RapidMiner` tool. Table 3.2 lists the integration of PMC with these external tools. Note that the last two integrations

Table 3.2: Tool integration with PMC.

Tool	Description
Log Explorer	Log visualizer plugin of ProM
Dotted Chart	Log visualizer/scatterplot plugin of ProM
Inductive Miner [102]	Process discovery plugin of ProM
Fuzzy Miner [64]	Process discovery plugin of ProM
Alignment-based Conformance checker [157]	Conformance checking plugin of ProM
Process Comparator	Process comparison tool of ProM (see Chapter 5)
RapidProM	Process mining workflow tool of RapidMiner (see Chapter 4)

correspond to contributions introduced in this thesis.

The process cubes created with PMC can also be exported and imported later. This is especially useful in situations where the cube has several complex dimensions.

The remainder of this section illustrates all the previously described features as a demonstration using real event data.

3.4 Applications

This section describes the usage of our tool and its interaction with other process mining techniques (e.g., process comparison, process mining workflows) through two step-by-step applications of PMC using the event data from the running example (see Section 2.4) related to a road fines management process.

3.4.1 Creating a Process Cube

This section describes how to create a process cube from scratch in PMC.

The first step in order to create a process cube in PMC is to import event data). PMC allows the import of event data in different formats (e.g., XES [1], CSV). When the event data is loaded, PMC automatically detects attribute types and value sets (see Figure 3.10). These can be modified by the user. After

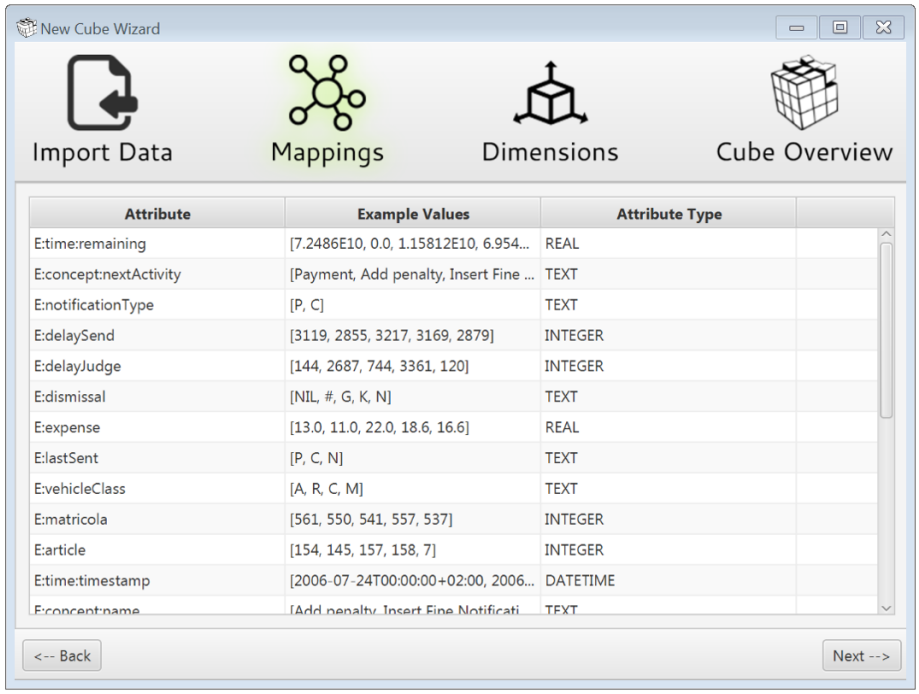


Figure 3.10: Importing event data into PMC.

this step, the user is prompted to create the dimensions of the cube, based on the event attributes obtained from the previous step, although, these can be modified. For each dimension, the user can specify the attributes that compose it by simply dragging them from the list of unused attributes and dropping them at the right dimension (see Figure 3.11). Note that PMC handles time in

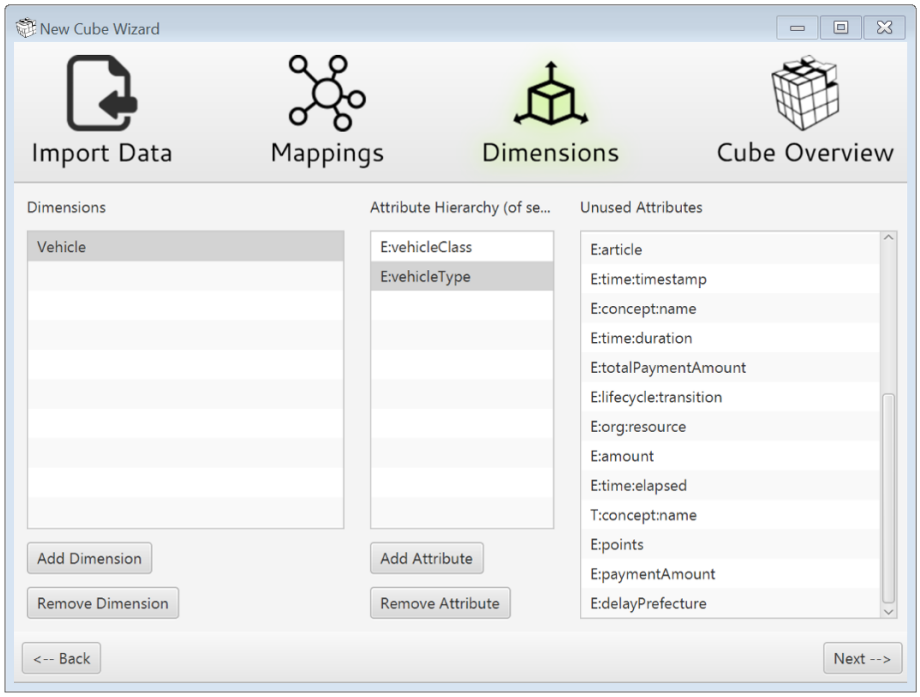


Figure 3.11: Defining dimensions of the cube and their attributes.

a specific manner. Given a *timestamp* event attribute, PMC builds a standard time dimension from it, including dimension attributes such as year, month, day, quartile, etc.

In a process cube with many dimensions, this can be an exhausting task, hence we provide a quick-setup option that creates a different dimension for each event attribute (each dimension contains only such attribute). After this step, PMC presents an overview summary of the cube, and the user gets to select a name and location to store the cube. Finally, the cube is stored in disk and it

is loaded into PMC and is now available for usage. Alternatively, the user can save the cube or load previously saved cubes.

3.4.2 Using a Process Cube

After opening PMC, and either loading or creating a process cube, the tool shows a list of the available cubes (see Figure 3.12). Once the user has selected a

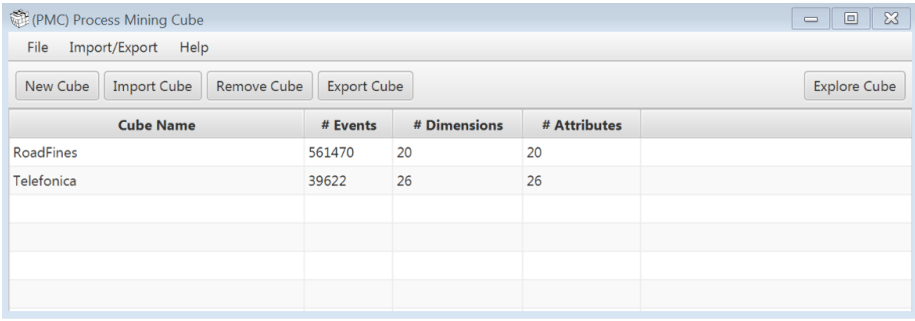


Figure 3.12: List of available process cubes in PMC.

process cube, the process cube explorer is shown as illustrated in Figure 3.13. The user interface of the process cube explorer is divided into two main parts:

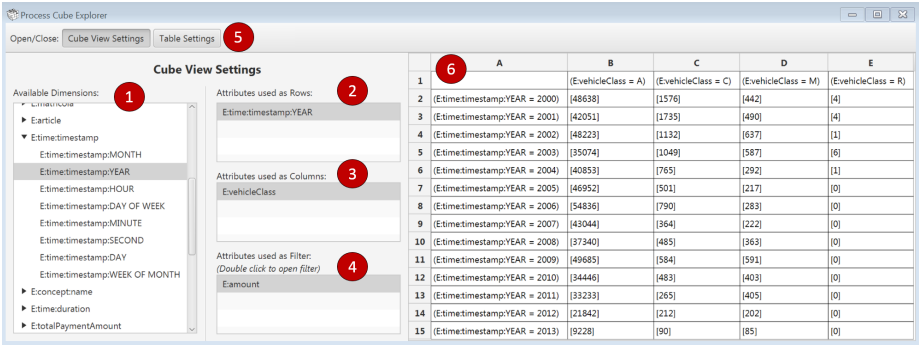
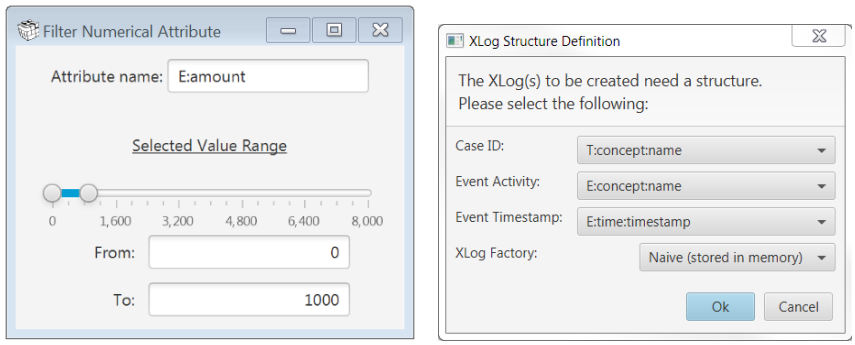


Figure 3.13: User interface of the Process Cube Explorer.

the *process cube view settings* and the *materialized cube cells*.



(a) Filter configuration in PMC (e.g., for slicing).

(b) Defining a Log structure for the events in the selected cell(s).

Figure 3.14: Configuration popups in PMC.

The *first* part (i.e., process cube view settings) defines the visible part of the cube as discussed in Section 3.2.1 and is divided into 4 elements.

The available dimensions panel (i.e., element (1) in Figure 3.13) represents the process cube structure (i.e., the available dimensions of the cube and their attributes), also as discussed in Section 3.2.1.

The available dimension attributes can be used to define the rows or columns of the cube, by dragging them into the corresponding list (i.e., elements (2) or (3) in Figure 3.13). The result of this is that the cube is *diced* over the dimension that contains such attribute, and the granularity of the dimension is set to such attribute. For example, in Figure 3.13, we diced the cube over the *Time* dimension (where the granularity is set to `Year`) which is represented in the resulting rows of the cube. Additionally, we diced the cube over the *Vehicle* dimension (where the granularity is set to `Vehicle Class`) which is represented in the resulting columns of the cube.

One can also slice dimensions in PMC. This is done by dragging dimension attributes into the *filter* list (i.e., element (4) in Figure 3.13). The result of this is that the cube is *sliced* over the dimension that contains such attribute. This means that the sliced dimension will not be used to define the cells of the cube, but will be used to determine which events will be used by the cube. PMC allows the user to modify the selection of values for the sliced dimensions by simply double clicking them, as illustrated in Figure 3.14a. A customized popup will emerge and the user can select the values that he/she wants to keep. This

is useful in scenarios where the events have to be filtered. For example, in Figure 3.13 and Figure 3.14a, we sliced the cube over the *Amount* dimension, keeping only the events with an amount equal or lower than 1000 euros.

The *second* part of the user interface (i.e., element (6) in Figure 3.13) corresponds to the resulting cells of the cube, filled with events from the event base (see Section 3.2.3).

A cell in a process cube is related to a set of events. In PMC, cells are also related to *numerical metrics* that can provide insights about their underlying set of events. Examples of such metrics are the number of events in the cell, the number of cases in the cell (after the set of events is transformed into an event log), the average size of cases in the cell, the average duration of cases in the cell, etc. All the metrics mentioned above are implemented in PMC, and can be configured by the user in the *table settings* panel (i.e., element (5) in Figure 3.13).

Once a materialized process cube view is produced (see Section 3.2.3) the contents of the materialized cells can be used by many process mining techniques. Often, process mining techniques require an event log as input, and since PMC produces sets of events as the output of each cell, such sets need to be converted into an event log in order to be used by external process mining techniques. In PMC this is done through a popup that appears as illustrated in Figure 3.14b when such conversion is needed.

3.4.3 Interaction with other Process Mining Techniques

After the contents of the selected cells have been converted to event logs, they can be used by many process mining plugins connected to PMC through its integration with ProM and RapidProM. For example, Figure 3.15 shows a selected cell transformed into an event log and visualized using the *Log Explorer*.

Figure 3.16 shows a selected cell transformed into an event log and then a Petri net is discovered from it using the "infrequent" version of the *Inductive Miner* [102] with a noise threshold of 0.2 (i.e., the *default* parameters).

Cells can be checked for conformance, as shown in Figure 3.17. A group of cells is selected, and their corresponding event logs can be merged into a single event log that is used for discovering a process model. This process model is then checked for conformance with respect to the event log used to discover it.

Event logs related to different cells can also be compared against each other. Figure 3.18 shows the results of such comparison using the process comparison tool (described in detail in Chapter 5).

Figure 3.15: The events in the selected cells can be visualized with the Log Visualizer plugin from ProM.

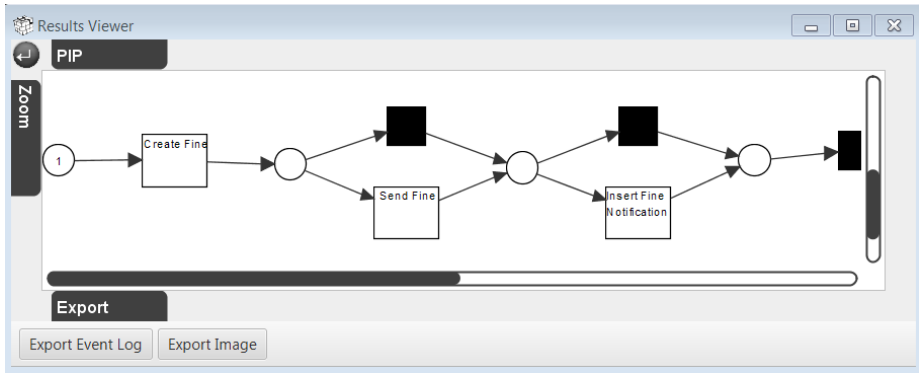


Figure 3.16: A process model can be discovered from the events in the selected cells.

Cells can also be used as input for process mining workflows. This is explained in more detail in Chapter 4. Figure 3.19 shows how a process mining workflow (described in Chapter 4) can be executed once for each cell, or only once for all the cells combined. The first time that a workflow is executed, PMC will request for the installation folder of RapidMiner, as this is necessary to boot RapidMiner in the background. After this, the workflow file is requested, and is later executed for the selected cells.

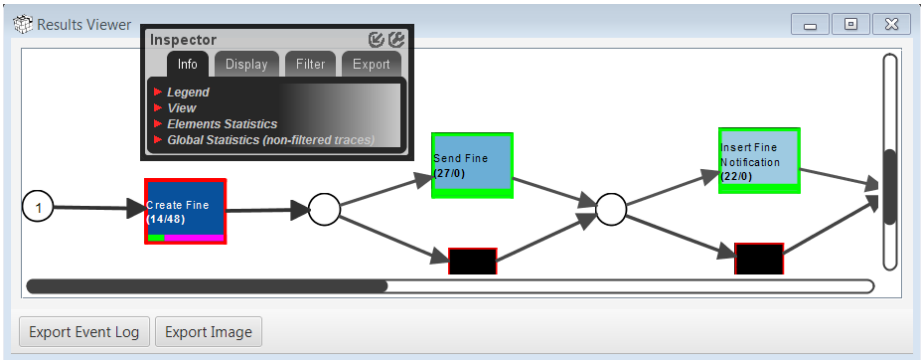


Figure 3.17: Event logs related to selected cells can be checked for conformance with respect to the process model discovered from the combined event logs of such selected cells.

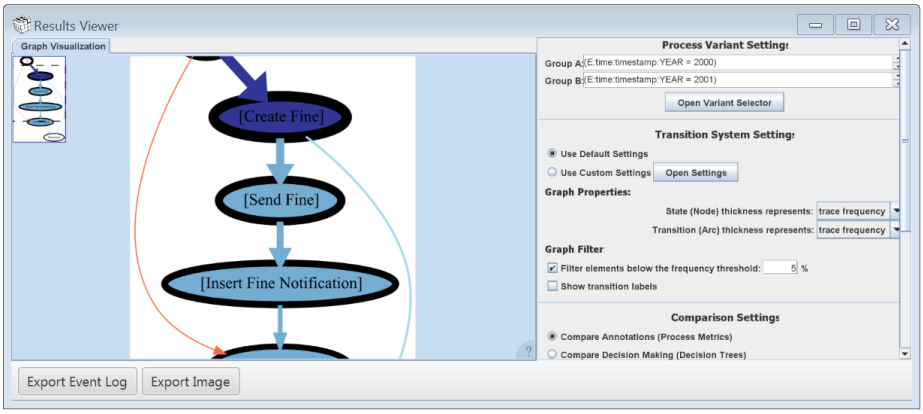


Figure 3.18: Event logs related to different cells can be compared using the Process Comparator plugin of ProM.

Going back to Figure 3.13, a quick glance at this set of cells shows that most of the fines are related to cars (A), and that the fines related to trucks (C) have decreased significantly over the years. Note that the numbers on the cells indicate the number of events in it. Also note that the fines related to trailers (R) only occurred between the years 2000 and 2004.

In Figure 3.20, the *Inductive Miner* [102] was used to discover a model from

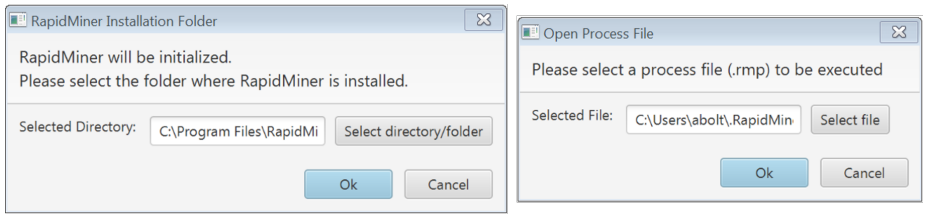


Figure 3.19: Process mining workflows can be executed using the events contained in the selected cell(s) as input.

each of two selected cells i.e., events that happened in the year 2011 related to trucks (C) and motorbikes (M) (i.e., cells C13 and D13 in Figure 3.13 respectively). Note that the process models discovered from these two cells certainly

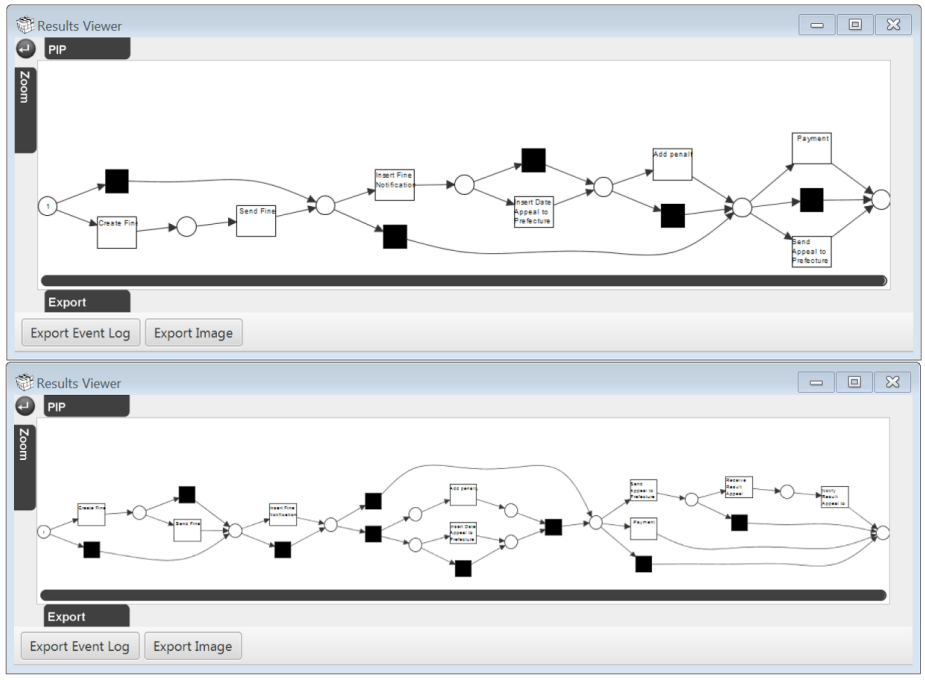


Figure 3.20: Process models discovered from events related to fines involving trucks (up) and motorcycles (down) in the year 2011.

have some differences. In a similar way, many other process mining techniques can be used instead.

In Figure 3.21, the same cube was now *sliced* over the elapsed time of events: all events that have an elapsed time within the trace of a week or more are removed. We do this by simply adding the *elapsed time* attribute to the filters list, and then selecting the value range that we want to keep. In other words,

	A	B	C	D	E
1		(E:vehicleClass = A)	(E:vehicleClass = C)	(E:vehicleClass = M)	(E:vehicleClass = R)
2	(E:time:timestamp:YEAR = 2000)	[4]	[4]	[4]	[5]
3	(E:time:timestamp:YEAR = 2001)	[4]	[4]	[4]	[4]
4	(E:time:timestamp:YEAR = 2002)	[3]	[3]	[4]	[0]
5	(E:time:timestamp:YEAR = 2003)	[4]	[4]	[4]	[3]
6	(E:time:timestamp:YEAR = 2004)	[3]	[4]	[4]	[0]
7	(E:time:timestamp:YEAR = 2005)	[3]	[4]	[4]	[0]
8	(E:time:timestamp:YEAR = 2006)	[3]	[4]	[4]	[0]
9	(E:time:timestamp:YEAR = 2007)	[3]	[4]	[4]	[0]
10	(E:time:timestamp:YEAR = 2008)	[3]	[3]	[4]	[0]
11	(E:time:timestamp:YEAR = 2009)	[3]	[3]	[4]	[0]
12	(E:time:timestamp:YEAR = 2010)	[3]	[3]	[3]	[0]
13	(E:time:timestamp:YEAR = 2011)	[3]	[3]	[4]	[0]
14	(E:time:timestamp:YEAR = 2012)	[2]	[3]	[4]	[0]
15	(E:time:timestamp:YEAR = 2013)	[2]	[2]	[2]	[0]

Figure 3.21: Materialized Process Cube view obtained by dicing the *Time* and *Vehicle Class* dimensions, and removing all events that happened a week after the start of each case. The number in each cell represents the average case size in terms of the number of events.

the cube view shows only the events that happened in the first week of a case since it started. The rows and columns are the same as before, but the number in each cell now represents the average size of the cases (in terms of number of events) after the events are filtered. Note that for trucks (C) initially in the early 2000's four events occurred in the first week of each case (in average). By the end of 2013 this was reduced to half. This suggests that at least in the first week, the process is getting slower, as in the last year only two events could be executed in the first week of each fine.

3.5 Conclusions

As process mining techniques are maturing and more event data becomes available, we no longer want to restrict analysis to a single all-in-one process. We would like to analyze and compare different variants (behaviors) of the process from different perspectives. Organizations are interested in comparative process mining to see how processes can be improved by understanding differences between groups of cases, departments, etc. We propose to use *process cubes* as a way to organize, split, and explore event data in a *multi-dimensional data structure tailored towards process mining* by splitting such event data using different data *dimensions* (i.e., given in the event data or derived from it) into process variants in a way that can help exposing differences between such variants, e.g., by using process comparison techniques.

This chapter extends the formalization of process cubes proposed in [155], providing a working implementation with an adequate performance needed to conduct multidimensional analysis using large event sets. The new framework gives end users the opportunity to analyze, explore and compare processes interactively on the basis of a multidimensional view on event data. We implemented the ideas proposed in this chapter in our *PMC* tool, and we encourage the process mining community to use it. There is a huge interest in tools supporting process cubes and the practical relevance is obvious.

Chapter 4

Process Mining Workflows

Scientific Workflow Management (SWFM) systems help users to design, compose, execute, archive, and share workflows that represent some type of analysis or experiment. Scientific workflows are often represented as directed graphs where the nodes represent "work" and the edges represent paths along which data and results can flow between nodes. Next to "classical" SWFM systems such as Taverna [71] and Kepler [108], one can also see the uptake of integrated environments for data mining, predictive analytics, business analytics, machine learning, text mining, reporting, etc. Notable examples are RapidMiner [66] and KNIME [9]. These can be viewed as SWFM systems tailored towards the needs of data scientists.

In process mining, typically many analysis steps need to be chained together i.e., a *process mining workflows*. Existing process mining tools do not support such analysis workflows. As a result, analysis may be tedious and it is easy to make errors. Repeatability and provenance are jeopardized by manually executing more involved process mining workflows.

A process mining workflow is basically composed of building blocks (i.e., analysis steps) that are chained together so that the output produced by the building blocks can be used as input by other building blocks. The approach presented in this chapter has been implemented based on building blocks obtained from the process mining framework *ProM* and the workflow and data mining capabilities of *RapidMiner*. The resulting tool is called *RapidProM* which explicitly supports process mining workflows. Overall, RapidProM offers a comprehensive support for any type of analysis involving event data and processes.

Figure 4.1 illustrates the integration between process cubes and process mining workflows. A process mining workflow tool (i.e., RapidProM) may take event data from different sources (e.g., event logs, cells of a *process cube* defined in Chapter 3, databases, smart devices) as input for executing process mining workflows and can produce both process-mining and non-process-mining results e.g., process models, filtered event logs, reports, charts, etc.

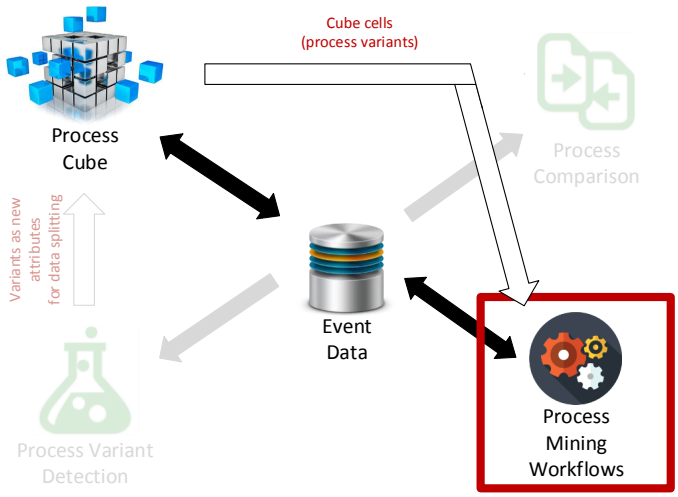


Figure 4.1: Overview of the scope of this chapter: a process mining workflow takes event data as input (either directly from event logs, or from the cells of a process cube) and executes process mining and non-process mining analysis steps in a designed workflow in order to produce results such as process models, reports, etc.

Since RapidProM is a part (i.e., an extension) of the *RapidMiner* suite, the process mining building blocks provided by RapidProM can be integrated and combined with pre-existing non-process-specific data mining building blocks related to data cleansing, filtering, preprocessing steps, and many others, and can leverage on the functionalities that analytic workflow tools can provide. Moreover, established data mining analysis (e.g., filtering, clustering and classification) can also be incorporated in such workflows in order to combine process mining with other types of analysis (see Challenge “Combining Process Mining

With Other Types of Analysis" in [152]).

This chapter is structured as follows. Section 4.1 discusses related work and positions our contribution. Section 4.2 describes the types of building blocks that are commonly used to construct a process mining workflow. Section 4.3 describes RapidProM, a tool supporting scientific workflows for process mining. Section 4.4 describes the application of process mining workflows in many use cases in process mining where the use of process mining workflows is crucial. It also discusses the interaction of process mining workflows with the *process cubes* presented in Chapter 3. Finally, Section 4.5 concludes the chapter.

4.1 Related Work

Conventional Business Process Management (BPM) [50] and Workflow Management (WfM) [104, 165] approaches and tools are mostly model-driven with little consideration for event data analysis. They are simply not designed for handling scientific workflows. On the other hand, Data Mining (DM) [65], Business Intelligence (BI), and Machine Learning (ML) [118] focus on data without considering end-to-end process models.

This chapter takes a *different* perspective on the gap between data analytics and BPM/WfM. We propose to use workflow technology for process mining rather than the other way around, namely, applying process mining to provide insights and, hence, advantages for workflow technologies. To this end, we focus on particular kinds of scientific workflows composed of process mining operators.

Differences between scientific and business workflows have been discussed in literature [7]. Despite unification attempts (e.g., [141]) both domains have remained quite disparate due to differences in functional requirements, selected priorities, and disjoint communities.

Naturally, the work reported in this chapter is closer to scientific workflows than business workflows (i.e., traditional BPM/WFM from the business domain). Numerous *Scientific Workflow Management (SWFM) systems* have been developed. Examples include Taverna [71], Kepler [108], Galaxy [60], ClowdFlows [96], jABC [142], Vistrails, Pegasus, Swift, e-BioFlow, VIEW, and many others. Some of the SWFM systems (e.g., Kepler and Galaxy) also provide repositories of models. The website `myExperiment.org` lists over 3500 workflows shared by its members [59]. The diversity of the different approaches illustrates that the field is evolving in many different ways. We refer to [147] for an extensive introduction to SWFM. Most of the scientific workflow management

systems (with the exception of the RapidProM extension of the analytic workflow suite RapidMiner) do not support process mining. Yet, process models and event logs are very different from the artifacts typically considered.

An approach to mine process models for scientific workflows (including data and control dependencies) was presented in [194]. Instead, this approach "uses process mining for scientific workflows" rather than applying scientific workflow technology to process mining. The results in [194] can be used to recommend scientific workflow compositions based on actual usage.

There are many approaches that aim to analyze repositories of scientific workflows. In [191], the authors provide an extensible process library for analyzing jABC workflows empirically. In [49] graph clustering is used to discover subworkflows from a repository of workflows. Other analysis approaches include [57], [106], and [186].

Explicit tool support for process mining workflows is still limited: none of the existing process mining tools (ProM [172], Disco, Perceptive, Celonis, QPR, Apromore [100], etc.) provides a complete and explicit set of functionalities to easily design and execute complex analysis workflows. Recently, PM4Py [10] and bupaR [77] have been introduced as process mining libraries (in python and R respectively) that can be used to write *scripts* that execute chains of process mining steps. However, scripting can be viewed as UI-less primitive workflow support, as it lacks typical scientific workflow functionalities and requires intensive knowledge of a programming language in order to be used. Furthermore, scripting becomes cumbersome and error-prone for complex workflows.

To our knowledge, RapidProM is the only approach explicitly supporting "scientific workflows for process mining". In the demo paper [115], Mans reported on the first implementation. In the meantime, RapidProM has been refactored based on various practical experiences.

Scientific workflows have been developed and adopted in various disciplines, including physics, astronomy, bioinformatics, neuroscience, earth science, economics, health, and social sciences. Various collections of reusable workflows have been proposed for all of these disciplines. For example, in [150] the authors describe workflows for quantitative data analysis in the social sciences.

The boundary between data analytics tools and scientific workflow management systems is not well-defined. Tools like RapidMiner [66] and KNIME [9] provide graphical workflow modeling and execution capabilities.

According to Gartner's 2017 magic quadrant for data science platforms, RapidMiner is one of the current leaders in the market (see Figure 4.2).[1] As

[1]Gartner's 2017 magic quadrant for data science platforms report can be found here: `https:`

Figure 4.2: Gartner's Magic Quadrant for Data Science Platforms (February 2017).

claimed in the report, platforms such as RapidMiner and KNIME have good user interfaces, tool support and exhaustive sets of functionalities. In this report, Gartner also elaborates on the pros and cons of each platform considering several aspects. The choice to focus on RapidMiner instead of KNIME is based on the report by Gartner, where it is stated that:

> "KNIME's platform can be difficult to scale and share across enterprise deployments. Several survey respondents indicated that their main challenge with KNIME is slow performance or an inability to build and deploy models in the needed time frame. Other issues with performance and scalability could reflect a need for additional components beyond the KNIME Analytics Platform."

This performance difference, and the fact that KNIME is slightly behind RapidMiner in the market (see Figure 4.2) lead us to choose RapidMiner as our plat-

`//www.gartner.com/doc/3606026/magic-quadrant-data-science-platforms`

form to work on.

4.2 Process Mining Workflows

In order to create scientific workflows for process mining, the building blocks that compose them need to be defined. This section discusses a taxonomy of such building blocks inspired by the so-called "BPM use cases" introduced in [153].

The *Process Mining Building Blocks* (PMBB) are characterized by two main aspects. First, they are *abstract* as they are not linked to any specific technique or algorithm. Second, they represent logical units of work, i.e., they cannot be conceptually split while maintaining their generality. This does not imply that concrete techniques that implement process-mining building blocks cannot be composed by micro-steps, according to the implementation and design that was used.

A process mining building block describes the objective and purpose of the step, and can be implemented by many techniques and algorithms. For example, the building block "*discover a process model from event data*" can be implemented by many process discovery techniques such as the Alpha miner [167], the Inductive miner [102] and many others. The process mining building blocks proposed in this chapter are grouped into six categories based on their purpose:

- *Event data extraction:* Building blocks to extract data from systems or to create synthetic data.
- *Event data transformation:* Building blocks to pre-process data (e.g., splitting, merging, filtering, and enriching) before analysis.
- *Process model extraction:* Building blocks to obtain process models through e.g., discovery, selection from a repository, or via process model generation tools.
- *Process model and event analysis:* Building blocks to evaluate event logs and models, e.g., to check the internal consistency or to check conformance with respect to an event log.
- *Process model transformations:* Building blocks to repair, merge or decompose process models.

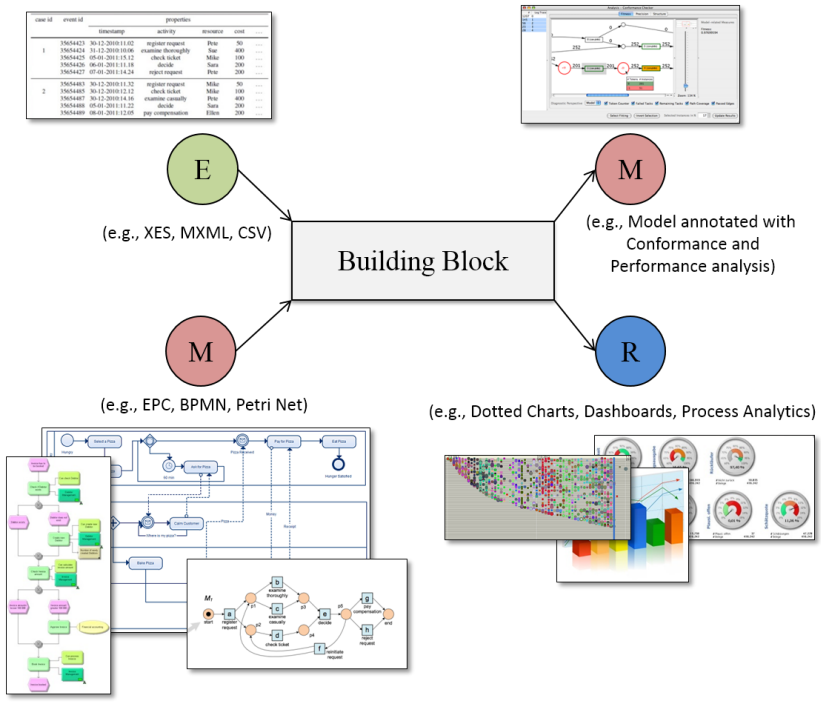


Figure 4.3: Generic example of a Building Block transforming a process model (M) and event data (E) into process analytics results (R) and an annotated process model (M).

- *Process model enhancement:* Building blocks to enrich event logs with additional perspectives or to suggest process improvements.

Each process mining building block takes a number of inputs and produces certain outputs, as illustrated in Figure 4.3. The input elements represent the set (or sets) of abstract objects required to perform the operation. The process mining building block component represents the logical unit of work needed to process the inputs and produce the outputs. Inputs and outputs are indicated through *circles* whereas a process-mining building block is represented by a *rectangle*. Arcs are used to connect the blocks to the inputs and outputs.

Two process-mining building blocks a and b are *chained* if one or more outputs of a are used as an inputs in b. As mentioned before, inputs and outputs are

depicted by circles. The letter inside a circle, and the color of the circle specify the type of the input or output. The following types of inputs and outputs are considered:

- (M) *Process models*, which are a representation of the behavior of a process, are represented by the letter `M` and are colored red. Here we abstract from the notation used, e.g., Petri nets, Heuristics nets, BPMN models are concrete representation languages.
- (E) *Event data sets*, which contain the recording of the execution of process instances within the information system(s), regardless of the format. They are represented by letter `E` and are colored green.
- (S) *Information systems*, which supports the performance of processes at runtime. They are represented by the letter `S` and are colored white. Information systems may generate events used for analysis and process mining results (e.g., prediction) may influence the information system.
- (P) *Sets of parameters* to configure the application of process-mining building blocks (e.g., thresholds, weights, ratios, etc.). They are represented by the letter `P` and are colored yellow.
- (R) *Results* that are generated as outputs of a process-mining building blocks. This can be as simple as a number or more complex structures like a detailed report. In principle, the types enumerated above in this list (e.g., process models) can also be results. However, it is worth to differentiate those specific types of outputs from results which are not process mining specific (e.g., a bar chart). Results are represented by the letter `R` and are colored blue.
- (D) *Additional Data Sets* that can be used as input for certain process-mining building blocks. These are represented by the letter `D` and are colored black. Such an additional data set can be used to complement event data with context information (e.g., one can use weather or stock-market data to augment the event log with additional data).

A *process mining workflow* is a chain of process mining building blocks in the form of a directed bigraph, which is defined by a set of *building blocks*, a set of *inputs and outputs* and a set of directed arcs that connect building blocks to inputs and outputs (and vice versa) and defines the *data flow* of the process mining workflow.

Process mining workflows are not necessarily sequential: some steps may be executed under specific circumstances only and some steps may be executed in parallel or multiple times (i.e., loops).

The remainder of this section provides a taxonomy of process-mining building blocks grouped into the six categories described before. For each category, several building blocks are provided. They were selected because of their usefulness for the definition of many process-mining scientific workflows. The taxonomy is not intended to be exhaustive; there will be new process-mining building blocks as the discipline evolves. Section 4.3 discusses how these building blocks can be implemented into concrete operators and provides examples of these operators implemented in RapidProM.

4.2.1 Event Data Extraction

Event data are the *cornerstone* of process mining. In order to be used for analysis, event data has to be extracted and made available. All of the process-mining building blocks of this category can extract event data from different sources. Figure 4.4 shows some process-mining building blocks that belong to this category.

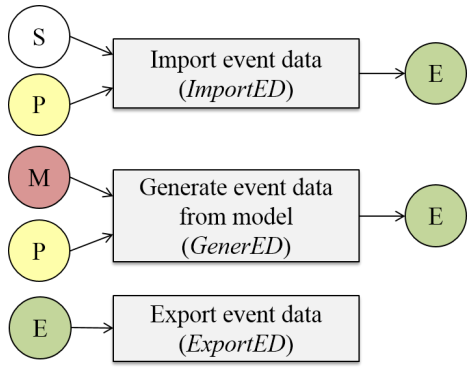


Figure 4.4: Process-mining building blocks related to event data extraction.

Import event data (*ImportED*) Information systems store event data in different formats and media, from files in a hard drive to databases in the cloud. This building block represents the functionality of extracting event data

from any of these sources. Some parameters can be set to drive the event-data extraction. For example, event data can be extracted from files in standard formats, such as XES, or from transactional databases.

Generate event data from model (*GenerED*) In a number of cases, one wants to assess whether a certain technique returns the expected or desired output (i.e., synthetic event data). For this assessment, controlled experiments are necessary where input data is generated in a way that the expected output of the technique is clearly known. Given a process model M, this building block represents the functionality of generating event data that records the possible execution of instances of M. This is an important function, e.g., for testing a new discovery technique. Many event simulators have been developed to support the generation of event data.

Export event data (*ExportED*) In many analysis situations, event data needs to be exported and stored for later use. For example, simulated or generated event data needs to be stored in order to be used later, e.g., for double-checking purposes.

4.2.2 Event Data Transformation

Sometimes, event data sets are not sufficiently rich to enable certain process mining analysis. In addition, certain data set portions should be excluded, because they are irrelevant, out of the scope of the analysis or even noise. Therefore, a number of event data transformations may be required before doing further analysis. This category comprises the building blocks to provide functionalities to perform the necessary event data transformations. Figure 4.5 shows the repertoire of process-mining building blocks that belong to this category.

Add data to event data (*AddED*) In order to perform a certain analysis or to improve the results, the event data can be augmented with additional data coming from different sources. For instance, if the process involves citizens, the event data can be augmented with data from the municipality data source. If the level of performance of a process is suspected to be influenced by the weather, event data can incorporate weather data coming from a system storing such a kind of data. If the event data contain ZIP codes, then other data fields such as country or city can be added to the event data from external data sources. This building block represents the functionality of augmenting event data using external data, represented as a generic data set in the figure.

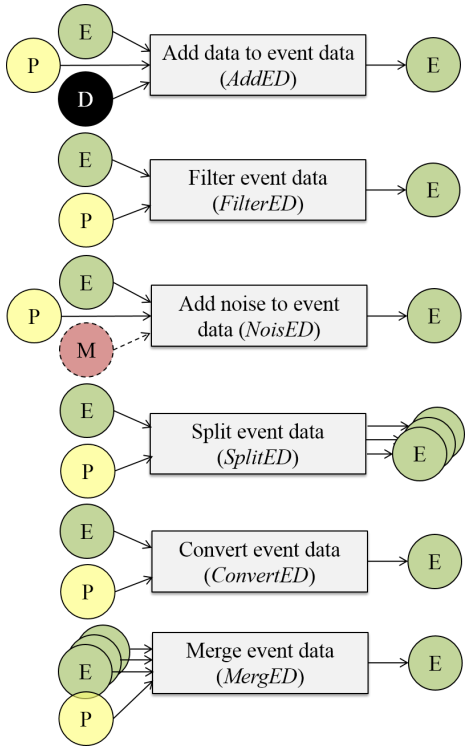


Figure 4.5: Process-mining building blocks related to event data transformations

Filter event data (*FilterED*) Several reasons may exist to filter out part of the event data. For instance, the process behavior may exhibit *concept drifts* over time. In those situations, the analysis needs to focus on certain parts of the event data instead of all of it. One could filter the event data and use only those events that occurred, e.g., in 2018. As a second example, the same process may run at different geographical locations. One may want to restrict the scope of the analysis to a specific location by filtering out the event data referring to different locations. This motivates the importance of being able to filter event data in various ways.

Add noise to event data (*NoisED*) All data is naturally subject to noise. In the case of event data, noise can be related to anomalous or infrequent process

behavior. In most settings, noise needs to be removed before analyzing the process (e.g., filtering). However, to study the effects of noise in event data, one needs to be able to add noise in a controlled manner in order to analyze its impact. Optionally, a reference process model can be used to steer the noise added to the data. This building block represents the functionality of adding noise to event data.

Split event data (*SplitED*) Sometimes, the organization generating the event data is interested in comparing the process' performances for different customers, offices, divisions, involved employees, etc. To perform such comparison, the event data needs to be split according to a certain criterion, e.g., according to organizational structures, and the analysis needs to be iterated over each portion of the event data. Finally, the results can be compared to highlight differences. Alternatively, the splitting of the data may be motivated by the size of the data. It may be intractable to analyze all data without decomposition or distribution. Many process-mining techniques are exponential in the number of different activities and linear in the size of the event log. If data is split in a proper way, the results of applying the techniques to the different portions can be fused into a single result. For instance, [154] discusses how to split event data while preserving the correctness of results. This building block represents the functionality of splitting event data into overlapping or non-overlapping portions.

Convert event data (*ConvertED*) Event data can be described and stored in many formats (e.g., XES, CSV). This building block represents the functionality of converting event data into a different format.

Merge event data (*MergED*) This process-mining building block is the inverse of the previous: data sets from different information systems are merged into a single event data set. This process-mining building block can also tackle the typical problems of data fusion, such as redundancy and inconsistency.

4.2.3 Process Model Extraction

Process mining revolves around process models to represent the behavior of a process. This category is concerned with providing building blocks to mine a process model from event data as well as to select or extract it from a process-

model collection. Figure 4.6 lists a number of process-mining building blocks belonging to this category.

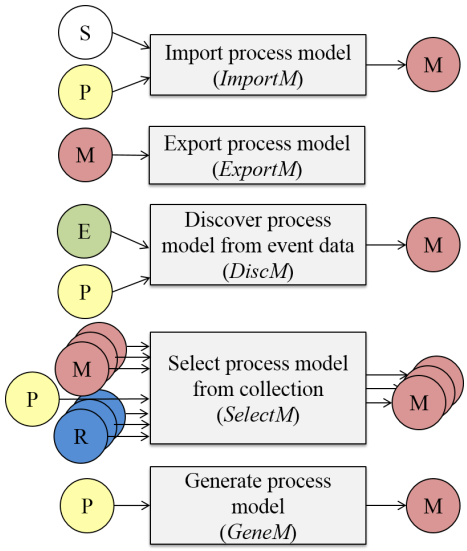


Figure 4.6: Process-mining building blocks related to process model extraction

Import process model (*ImportM*) Process models can be stored in some media for later retrieval to conduct some analyses. This building block represents the functionality of loading a process model from some repository.

Export process model (*ExportM*) Process models can be stored in some media for later retrieval to conduct some analyses. This building block represents the functionality of exporting a process model to be stored for future analysis.

Discover process model from event data (*DiscM*) Process models can be manually *designed* to provide a normative definition for a process. These models are usually intuitive and understandable, but they might not describe accurately what happens in reality. Event data represent the “real behavior" of the process. Discovery techniques can be used to mine a process model on the basis of the behavior observed in the event data [156]. Here, we stay independent of the specific notations and algorithms. Examples of

algorithms are the Alpha Miner [167], the Heuristics Miner [189] or, more recent techniques like the Inductive Miner [102]. This building block represents the functionality of discovering a process model from event data. This block, as many others, can receive a set of parameters as an input to customize the application of the algorithms.

Select process model from collection (*SelectM*) Organizations can be viewed as a collection of processes and resources that are interconnected to form a *process ecosystem*. This collection of processes can be managed and supported by different approaches, such as ARIS [133] or Apromore [100]. To conduct certain analyses, one needs to use some of these models and not the whole collection. In addition, one can give a criterion to retrieve a subset of the collection. This building block represents the functionality of selecting one or more process models from a process-model collection.

Generate process model (*GeneM*) Process models can be artificially generated without the use of event data on the basis of configuration parameters that control the distributions of the presence of control-flow constructs e.g., sequences, parallelism, loops. Examples of process model generators are [32] and [86].

4.2.4 Process Model and Event Analysis

Organizations normally use process models for the discussion, configuration, and implementation of processes. In recent years, many process mining techniques are also using process models for analysis. This category groups process-mining building blocks that can analyze process models or event logs and provide analysis results. Figure 4.7 shows some process-mining building blocks that belong to this category.

Analyze process model (*AnalyzeM*) Process models may contain a number of structural problems. For instance, the model may exhibit undesired deadlocks, activities that are never enabled for execution, variables that are used to drive decisions without previously taking on a value, etc. Several techniques have been designed to verify the soundness of process models against deadlocks and other problems [166]. This building block refers to design-time properties: the process model is analyzed without considering how the process instances are actually being executed. The checking of the conformance of the process model against real event data is covered by the next building block (*EvaluaM*). Undesired design-time properties

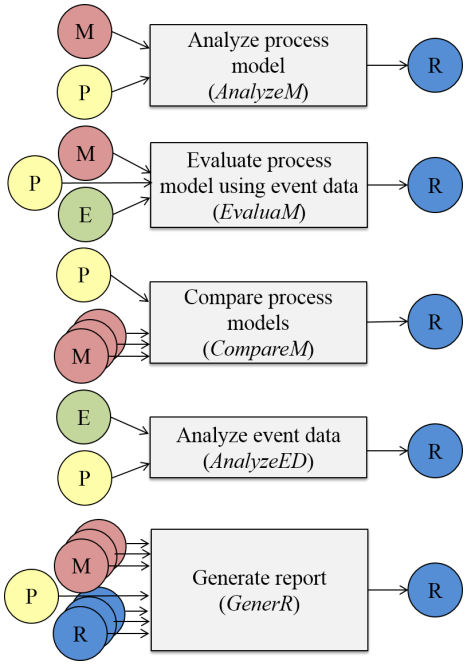


Figure 4.7: Process-mining building blocks related to process model and event analysis.

happen for models designed by hand but also for models automatically mined from event data. Indeed, several discovery techniques do not guarantee to mine process models without structural problems. This building block provides functionalities for analyzing process models and detecting structural problems.

Evaluate process model using event data (*EvaluaM*) Besides structural analysis, process models can also be analyzed against event data. Compared with the previous building block (*AnalyzeM*), this block is not concerned with a design-time analysis. Conversely, it makes a-posteriori analysis where the adherence of the process model is checked with respect to the event data, namely how the process has actually been executed. In this way, the expected or normative behavior as represented by the process model is checked against the actual behavior as recorded in the event data. In the literature, this is referred to as *conformance checking* [156].

This can be used, for example, in fraud or anomaly detection. Replaying event data on process models has many possible uses: Aligning observed behavior with modeled behavior is key in many applications. For example, after aligning event data and model, one can use the *time* and *resource* information contained in the log for performance analysis. This can be used for *bottleneck* identification or to gather information for simulation analysis or predictive techniques. This building block represents the functionality of analyzing or evaluating process models using event data.

Compare process models (*CompareM*) Processes are not static as they dynamically evolve and adapt to the business context and requirements. For example, processes can behave differently over different years, or at different locations. Such differences or similarities can be captured through the comparison of the corresponding process models. For example, the degree of similarity can be calculated. Approaches that explicitly represent configuration or variation points [159] directly benefit from such comparisons. Building block *CompareM* is often used in combination with *SplitED* that splits the event data into sublogs and *DiscM* that discovers a model per sublog.

Analyze event data (*AnalyzeED*) Instead of directly creating a process model from event data, one can also first inspect the data and look at basic statistics. Moreover, it often helps to simply visualize the data. For example, one can create a so-called dotted chart [156] exploiting the temporal dimension of event data. Every event is plotted in a two dimensional space where one dimension represents the time (absolute or relative) and the other dimension may be based on the case, resource, activity or any other property of the event. The color of the dot can be used as a third dimension. See [91] for other approaches combining visualization with other analytical techniques.

Generate report (*GenerR*) To consolidate process models and other results, one may create a structured report. The goal is not to create new analysis results, but to present the findings in an understandable and predictable manner. Generating standard reports helps to reduce the cognitive load and helps users to focus on the things that matter most.

4.2.5 Process Model Transformations

Process models can be designed or, alternatively, discovered from event data. Sometimes, these models need to be adjusted for follow-up analyses. This category groups process-mining building blocks that provide functionality to change the structure of a process model. Figure 4.8 shows some process-mining building blocks that belong to this category.

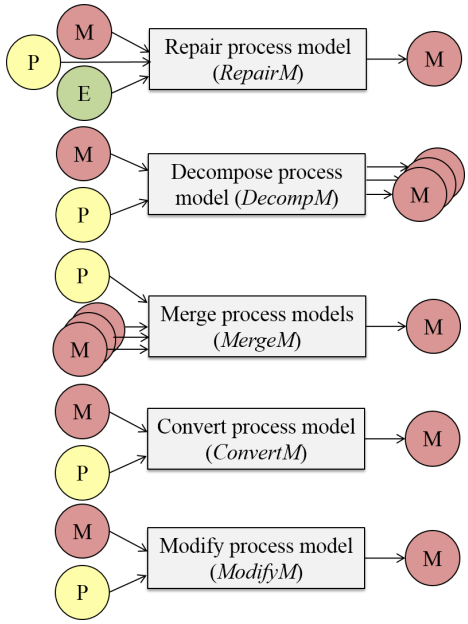


Figure 4.8: Process-mining building blocks related to process model transformations.

Repair process model (*RepairM*) Process models may need to be repaired in case of consistency or conformance problems. Repairing can be regarded from two perspectives: repairing structural problems and repairing behavioral problems. The first case is related to the fact that models can contain undesired design-time properties such as deadlocks and livelocks (see also the *Analyze process model* building block discussed in Section 4.2.4). Repairing involves modifying the model to avoid those properties. Techniques for repairing behavioral problems focus on models that are structurally sound but that allow for undesired behavior or behavior that does

not reflect reality. See also the *Evaluate process model using event data* building block discussed in Section 4.2.4, which is concerned with the discovery of conformance problems. This building block provides functionality for both types of repair.

Decompose process model (*DecompM*) Processes running within an organization may be extremely large, in terms of activities, resources, data variables, etc. As mentioned, many techniques are exponential in the number of activities. The computation may be improved by splitting the models into fragments, analogously to what was mentioned for splitting the event log. If the model is split according to certain criteria, the results can be somehow amalgamated and, hence, be meaningful for the entire model seen as a whole. For instance, the work on decomposed conformance checking [154] discusses how to split process model to make process mining possible with models with hundreds of elements (such as activities, resources, data variables), while preserving the correctness of certain results (e.g., the fraction of deviating cases does not change because of decomposition). This block provides functionalities for splitting process models into smaller fragments.

Merge process models (*MergeM*) Process models may also be created from the intersection (i.e. the common behavior) or union of other models. This building block provides functionalities for merging process models into a single process model. When process discovery is decomposed, the resulting models need to be merged into a single model.

Convert process model (*ConvertM*) Process models can be described and stored in many notations, as described in Section 2.3 e.g., Petri net, BPMN, process tree. This building block represents the functionality of converting a process model into a different notation.

Modify process model (*ModifyM*) Existing process models can be structurally modified (e.g., a choice can be removed, activities can be put in parallel branches). This block represent the functionality of modifying process models.

4.2.6 Process Model Enhancement

Process models just describing the control-flow are usually not the final result of process mining analysis. Process models can be enriched or improved using

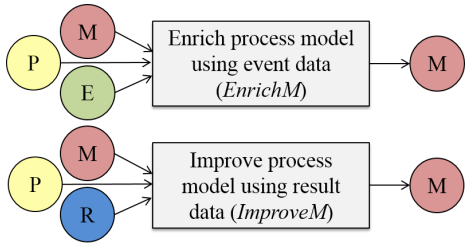


Figure 4.9: Process-mining building blocks related to process model enhancement.

additional data in order to provide better insights about the real process behavior that it represents. This category groups process-mining building blocks that are used to enhance process models. Figure 4.9 shows a summary of the process-mining building blocks that belong to this category.

Enrich process model using event data (*EnrichM*) The backbone of any process model contains basic structural information relating to control-flow. However, the backbone can be enriched with additional perspectives derived from event data to obtain better analysis results. For example, event frequency can be annotated in a process model in order to identify the most common paths followed by process instances. Timing information can also be used to enrich a process model in order to highlight bottlenecks or long waiting times. This enrichment does not have an effect on the structure of the process model. This building block represents the functionality of enriching process models with additional information contained in event data.

Improve process model (*ImproveM*) Besides being enriched with data, process models can also be improved. For example, performance data can be used to suggest structural modifications in order to improve the overall process performance. It is possible to automatically improve models using causal dependencies and observed performance. The impact of such modifications could be simulated in “what-if scenarios" using performance data obtained in the previous steps. This building block represents the functionality of improving process models using data from other analysis results.

4.3 Implementation

The framework to support process mining workflows proposed in this chapter has been implemented into RapidProM, which is an extension for RapidMiner [66] that offers several process mining techniques obtained from the process mining tool ProM [172]. The building blocks defined in Section 4.2 have been implemented in RapidProM as *Operators*. Most of the building blocks have been realized using RapidMiner-specific wrappers of plug-ins of the ProM Framework [172]. ProM is a framework that allows researchers to implement process mining algorithms in a standardized environment, which provides a number of facilities to support programmers. Nowadays, it has become the *de-facto* standard for process mining (see Section 1.1). The extension of RapidMiner to provide process-mining blocks for scientific workflows using ProM is also freely available. At the time of writing, RapidProM provides over 50 process mining operators, including several process-discovery algorithms and filters as well as importers and exporters from/to different process-modeling notations. The operators are defined as atomic steps, however, they can be composed into (sub) processes natively in RapidMiner. A (sub) process is the equivalent of a collapsed group of operators, but it can also be executed as an atomic block itself. This is allowed by RapidMiner's native concurrency management, which separates input from output object representations (i.e., a modified input does not affect any other parallel operators that use the same input).

The first version of RapidProM was presented during the BPM 2014 demo session [115]. This initial version successfully implemented around 30 basic process-mining functionalities and has been downloaded over 10.000 times since its release in July 2014 until its deprecation in November 2016. However, process mining is a relatively new discipline, which is developing and evolving rapidly. Therefore, various changes and extensions were needed to keep up with the state-of-the-art. The new version presented in this chapter incorporates implementations of various new algorithms, which did not exist in the first version, and also improved the performance and robustness of the algorithms [169]. This new version has been downloaded over 56.000 times since its release in November 2016 until September 2019.[2]

The RapidProM extension is hosted both at `http://www.rapidprom.org` and in the RapidProM extension manager server, which can be directly accessed through the RapidMiner Marketplace. After installation, the RapidProM opera-

[2]This figure only includes downloads using the RapidMiner Marketplace. Downloads and code usage from our server and its original source in GitHub are not considered.

tors are available for use in any RapidMiner workflow, which allows to combine process-mining with other data-mining techniques. Figure 4.10 shows an example of a process-mining scientific workflow implemented using RapidProM operators. Many of these operators implement a process-mining building block.

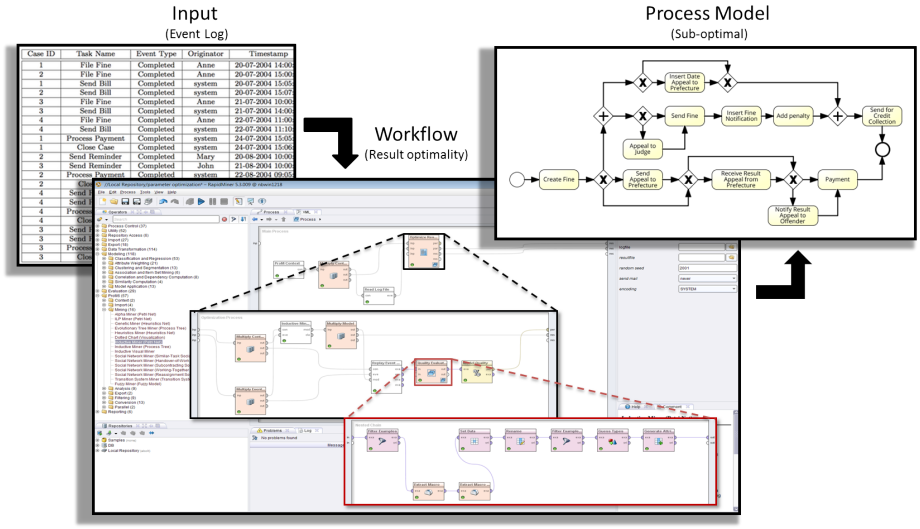


Figure 4.10: Example of a Process Mining Workflow in RapidMiner through the RapidProM extension: The workflow transforms Event data (Input) into a Sub-optimal Process Model (Output).

The process mining workflow shown in Figure 4.10 is used to obtain a sub-optimal process model from event data.

Readers are referred to `http://www.rapidprom.org` for detailed installation, setup and troubleshooting instructions.

This remainder of this section describes the available implementation of the building blocks described in Section 4.2 into RapidProM operators in Tables 4.1 to 4.6.

Table 4.1: Event Data Extraction Operators.

Building Block	Operator Name	Operator Description
ImportED	**Read Log (path)**	Imports an event log from a specified path
ImportED	**Read Log (file)**	Takes a file object (usually obtained from a "loop files" operator) and transforms it to an Event Log
ExportED	**Export Event Log**	Exports an Event Log in different formats
GenerED	**Generate Event Log from Process Tree** [86]	Generates an event log from a process model in a process tree format.

Table 4.2: Event Data Transformation Operators.

Building Block	Operator Name	Operator Description
AddED	**Add Table Column to Event Log**	Adds a single Data Table column as trace attribute to a given Event Log
AddED	**Add Trace Attributes to Event Log**	Adds all columns of a Data Table (except case id) as trace attributes to a given Event Log
AddED	**Add Event Attributes to Event Log**	Adds all columns of a Data Table (except case id and event id) as event attributes to a given Event Log
AddED	**Add Events to Event Log**	Adds Events to a given Event Log from selected columns on a Data Table
MergeED	**Merge Event Logs**	Merges two Event Logs
AddED	**Add Artificial Start and End Event**	Adds an artificial Start Event to the beginning, and an artificial End Event to the ending of each trace
AddED & *FilterED*	**Add Noise to Event Log**	Adds different types of noise (e.g., remove, add or swap activities) to the event log
ConvertED	**Event Log to ExampleSet**	Converts an Event Log into a Data Table (ExampleSet)
ConvertED	**ExampleSet to Event Log**	Converts a Data Table (ExampleSet) into an Event Log
SplitED	**Split Event Log**	Splits an event log into two non-overlapping sets of traces (i.e., event logs).

Table 4.3: Process Model Extraction Operators.

Building Block	Operator Name	Operator Description
ImportM	**Read PNML**	Imports a Petri Net in a *Petri Net Modeling Language* (PNML) format from a specified path
ExportM	**Export PNML**	Exports a Petri Net in PNML format
DiscM	**Alpha Miner** [167]	Discovers a Petri Net. Fast but results are not always reliable because of overfitting issues
DiscM	**ILP Miner** [170]	Discovers a Petri Net by solving *Integer Linear Programming* (ILP) problems. Result have perfect fitness but generally poor precision. Slow on large Logs
DiscM	**Evolutionary Tree Miner** [29]	Discovers a Process Tree using a guided genetic algorithms based on model quality dimensions. Guarantees soundness but cannot represent all possible behavior due to its block-structured nature
DiscM	**Heuristics Miner** [189]	Discovers a Heuristics Net using a probabilistic approach. Good when dealing with noise. Fast
DiscM	**Inductive Miner** [102]	Discovers a Process Tree or Petri Net. Good when dealing with infrequent behavior and large Logs. Soundness is guaranteed
DiscM	**Social Network Miner** [164]	Discovers a Social Network from the Event Log resources. Different Social Networks can be obtained: similar task, handover of work, etc.
DiscM	**Transition System Miner** [162]	Discovers a Transition System using parameters to simplify the space-state exploration.
DiscM	**Fuzzy Miner** [64]	Discovers a Fuzzy Model. Good when dealing with unstructured behavior. Fast
GeneM	**Generate Process Tree** [86]	Generates a process tree with defined control-flow structures.

Table 4.4: Process Model and Event Analysis Operators.

Building Block	Operator Name	Operator Description
AnalyzeED	**Dotted Chart** [140]	Shows the temporal distribution of events within traces
AnalyzeED	**Feature Prediction** [45]	Produces predictions of business process features using decision trees
AnalyzeM	**WOFLAN** [181]	Analyzes the soundness of a Petri Net using the Woflan software (`www.swmath.org/software/7028`)
AnalyzeM	**Select Fuzzy Instance**	Selects the best fuzzy instance from a Fuzzy Model
EvaluaM	**Measure the Precision of a Model** [3]	Measures the precision of a process model with respect to an event log.
EvaluaM	**Measure the Fitness of a Model** [2]	Measures the fitness of a process model with respect to an event log.

Table 4.5: Process Model Transformation Operators.

Building Block	Operator Name	Operator Description
RepairM	**Repair Model** [52]	Replays an Event Log in a Petri Net and repairs this net to improve fitness.
RepairM	**Reduce Silent Transitions**	Reduces a Petri Net by removing invisible transitions (and places) that are not used
ConvertM	**Reachability Graph to Petri Net**	Converts a Reachability Graph into a Petri Net
ConvertM	**Petri Net to Reachability Graph**	Converts a Petri Net into a Reachability Graph
ConvertM	**Heuristics Net to Petri Net**	Converts a Heuristics Net into a Petri Net
ConvertM	**Process Tree to Petri Net**	Converts a Process Tree into a Petri Net
ConvertM	**Petri Net to BPMN**	Converts a Petri Net into a BPMN model

Table 4.6: Process Model Enhancement Operators.

Building Block	Operator Name	Operator Description
EnrichM	**Inductive Visual Miner** [103]	Process exploration tool that shows an annotated interactive model for quick exploration of a Log
EnrichM	**Animate Log in Fuzzy Instance** [58]	Shows an animated replay of a Log projected over a Fuzzy Instance
EnrichM	**PomPom**	Petri Net visualizer that emphasizes those parts of the process that correspond to high-frequent events in a given Log
EnrichM	**Replay Log on Petri Net (Performance)** [157]	Replays a Log on a Petri Net and generates performance metrics such as throughput time, waiting time, etc.
EnrichM	**Replay Log on Petri Net (Conformance)** [157]	Replays a Log on a Petri Net and generates conformance metrics such as fitness

4.4 Applications

Building blocks can be chained together in many ways in order to support specific use cases. This section identifies generic use cases that are not domain-specific and, hence, that can be applied in different contexts. We provide concrete process mining workflows to support these use cases. The use cases are composed using the process mining building blocks defined before, and remain independent of any specific implementation of a building block. In fact, as mentioned before, the building blocks may employ different concrete techniques: there are dozens of process discovery techniques realizing instances of building block *DiscM* (see Section 4.2.3). This section considers five use cases that are made possible by process mining workflows:

1. *Result (sub-)optimality:* Often different process mining techniques can be applied and a priori it is not clear which one is most suitable. By modeling the analysis workflow, one can just perform all candidate techniques on the data, evaluate the different analysis results, and pick the result with the highest quality (e.g., the process model best describing the observed behavior). Note that results are regarded as sub-optimal. Optimality is not guaranteed in general because it depends on the techniques and parameters used.

2. *Parameter sensitivity:* Different parameter settings and alternative ways of filtering can have unexpected effects. Therefore, it is important to see how the quality of the results is sensitive to changes in these settings. It is important to not simply show the analysis result without having some confidence indications.

3. *Large-scale experiments:* Each year new process mining techniques become available and larger data sets need to be tackled. For example, novel discovery techniques need to be evaluated through massive testing and larger event logs need to be decomposed to make analysis feasible. Without automated workflow support, these experiments are tedious, error-prone and time consuming.

4. *Repeating questions:* Practical questions that can be answered through process mining analysis are often repetitive, e.g., the same analysis is done for a different period or a different group of cases. Process mining workflows facilitate recurring forms of analysis.

5. *Interaction with process cubes: The process mining workflows introduced in this chapter can interact with the* process cubes *introduced in Chapter 3): The cells of a process cube can be used as input for a process mining workflow, so the same workflow can be executed for each cell of the cube.*

Note that the same results could also be achieved without using scientific workflows. However, obtaining such results would require a tedious and error-prone work of repeating the same steps *ad nauseam*. The reminder of this section discusses the use cases described above, and shows concrete applications of them in which the usage of process mining workflows is crucial.

4.4.1 Result (Sub-)Optimality

This section discusses how process-mining building blocks can be used to mine optimal process models according to some optimality criteria. Often, in process discovery, optimality is difficult (or even impossible) to achieve. Often sub-optimal results are returned and it is not known what is "optimal".

Consider for example the process discovery task. The quality of a discovered process model is generally defined by four quality metrics [2, 3, 156, 157]:

- *Replay fitness:* quantifies the ability of the process model to reproduce the execution of process instances as recorded in event data.
- *Simplicity:* captures the degree of complexity of a process model, in terms of the numbers of activities, arcs, variables, gateways, etc.
- *Precision:* quantifies the degree with which the model allows for too much behavior compared to what was observed in the event data.
- *Generalization:* quantifies the degree with which the process model is capable to reproduce behavior that is not observed in the event data but that potentially should be allowed. This is linked to the fact that event data often are incomplete in the sense that only a fraction of the possible behaviors can be observed.

Traditionally, these values are normalized between 0 and 1, where 1 indicates the highest score and 0 the lowest.

The model of the highest value within a collection of (discovered) models is such that it can mediate among those criteria. Often, these criteria are in competing: higher score for one criterion may lower the score of a second criterion. For instance, in order to have a more precise model, it is necessary to

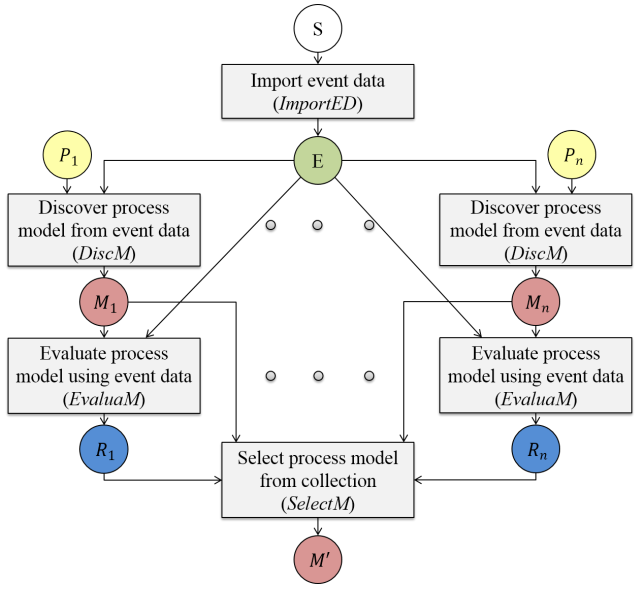


Figure 4.11: Result (sub-)optimality in process model discovery: process-mining scientific workflow for mining an optimal model in terms of a defined scoring criteria.

sacrifice the behavior observed in the event data that is less frequent, thus partly hampering the replay-fitness score.

Figure 4.11 shows a suitable generic scientific workflow for mining a process model from event data that is optimal with respect to a score defined by specific criteria. The optimization is done by finding the parameters that returns an optimal model.

Event data is loaded from an information system and used n times as input for a discovery technique using different parameter values. The n resulting process models are evaluated using the original event data and the model that has the best score is returned. Please note that the result is likely to be sub-optimal: n arbitrary parameter values are chosen out of a much larger set of possibilities. If n is sufficiently large, the result is sufficiently close to the optimal. This scientific workflow is still independent of the specific algorithm used for discovery; as such, the parameter settings are also generic.

Note that scientific workflows can also be hierarchically defined. For exam-

ple, the discover process-mining building block (*DiscM*) in Figure 4.11 can be replaced by an entire scientific sub-workflow.

To show the application of process mining workflows in this use case, we performed an experiment using RapidProM to extract the model that scores higher with respect to the harmonic average of precision and replay fitness i.e., the f1-score of fitness and precision.[3] The harmonic average of replay fitness and precision seems to be better than the arithmetic average since it is necessary to have a strong penalty if one of the criteria is low.

In this experiment, we employed the *Inductive Miner - Infrequent* discovery technique [102] and used different values for the *noise threshold* parameter. This parameter is defined in a range of values between 0 and 1. This parameter allows for filtering out infrequent behavior contained in event data in order to produce a simpler model: the lower the value is for this parameter (i.e., close to 0), the larger the fraction of behavior observed in the event data that the model allows. To measure fitness and precision, we employ the conformance-checking techniques reported in [2, 3].

We designed a process mining workflow where several models are discovered with different values of the *noise threshold* parameter. Finally, the workflow selects the model with the highest value of the harmonic average among those discovered. As input, we used the Road Fines running example (see Section 2.4).

All techniques used here are available as part of the RapidProM extension. A summary of the concrete operators used in this experiment for each building block is presented in Table 4.7.

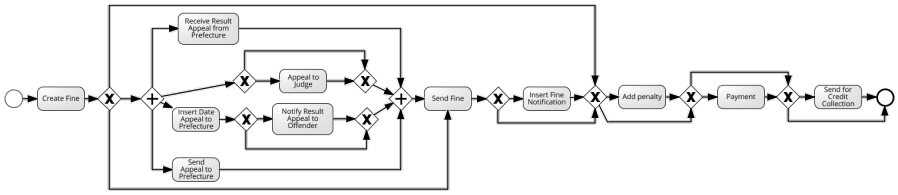
Table 4.7: Operators used in the Result (sub) Optimality experiment.

Building Block	Operator Name
ImportED	Read Log (file)
DiscM	Inductive Miner
EvaluaM	Replay Log on Petri Net
SelectM	Optimize Parameter (RapidMiner native operator)

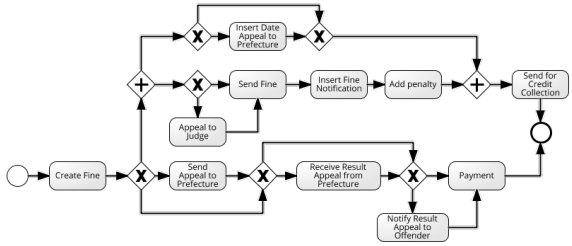
The result of this experiment is shown in Figure 4.12b, which corresponds to the model generated using the optimal parameters obtained through our

[3]The harmonic average of a and b used in this article is defined as 2*a*b/(a+b) and it is meant to penalize very low scores in either a or b.

scientific workflow, whereas Figure 4.12a illustrates the model generated using default parameters.



(a) Model mined using the Inductive Miner with default value of the noise-threshold parameter, which is 0.2. The harmonic average of fitness and precision is 0.708



(b) Model mined using the Inductive Miner with one of the best values of the noise-threshold parameter, which is 0.7. This value was obtained as a result of this experiment. The harmonic average of fitness and precision is 0.912

Figure 4.12: Comparison of process models that are mined with the default parameters and with the parameters that maximize the harmonic average of replay fitness and precision. The process is concerned with road-traffic fine management and models are represented using the BPMN notation.

There are clear differences between the models. For example, in the default model, parallel behavior dominates the beginning of the process. Instead, the "optimal model" presents simpler choices. Another example concerns the final part of the model. In the default model, the latest process activities can be skipped through. However, in the optimal model, this is not possible. The optimal model has a replay fitness and precision of 0.921 and 0.903 respectively, with harmonic average 0.912. It scores better than the model obtained through default parameters, where the replay fitness and precision is 1 and 0.548, respectively, with harmonic average 0.708. The optimal model was generated

with value 0.7 for the noise threshold parameter.

4.4.2 Parameter Sensitivity

Parameters are used by techniques to customize their behavior, e.g., adapting to the noise level in the event log. These parameters have different ways of affecting the results produced, depending on the specific design of the technique or algorithm. Some parameters can have more *relevance* than others (i.e., they have a more substantial effect on the results). There are many ways to evaluate the sensitivity of a certain parameter for a given algorithm. Figure 4.13 shows an example of this use case. Here the parameter value P is varied (i.e., from P_1 to P_Δ). For each of the discovered models, a score is computed. The results are finally plotted on a Cartesian coordinate system where the X-axis is associated with the parameter's values and the Y-axis is associated with the model's score.

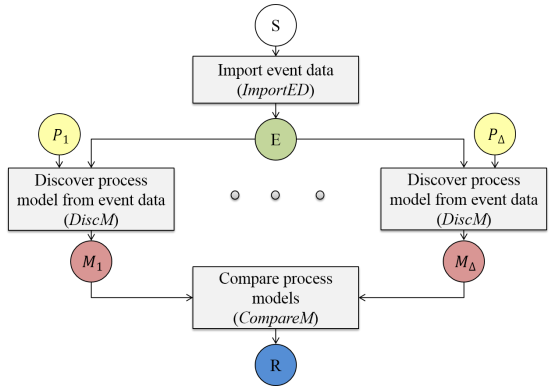


Figure 4.13: Parameter sensitivity in process discovery techniques: process mining workflow for comparing the effects of different parameter values for a given discovery technique

To show an application of this use case, we conducted an experiment that analyses the sensitivity of the *noise threshold* parameter of the *Inductive Miner - infrequent* [102]. We used again the event data of the Road Fine management process (see Section 2.4)

A summary of the concrete operators used in this for each building block is presented in Table 4.8.

Table 4.8: Operators used in the Parameter Sensitivity experiment.

Building Block	Operator Name
ImportED	Read Log (file)
DiscM	Inductive Miner
EnrichM	Replay Event Log on Petri Net, Create chart from value array (RapidMiner native operator)

In this experiment, we implemented a process mining workflow using RapidProM to explore the effect of this parameter in the final quality of the produced model. In order to do so, we discovered 41 models using different parameter values between 0 and 1 (i.e., a step-size 0.025) and evaluated their quality through the harmonic average of replay fitness and precision used before.

Figure 4.14 shows the results of these evaluations, showing the variation of the harmonic average for different values of the *noise threshold* parameter.

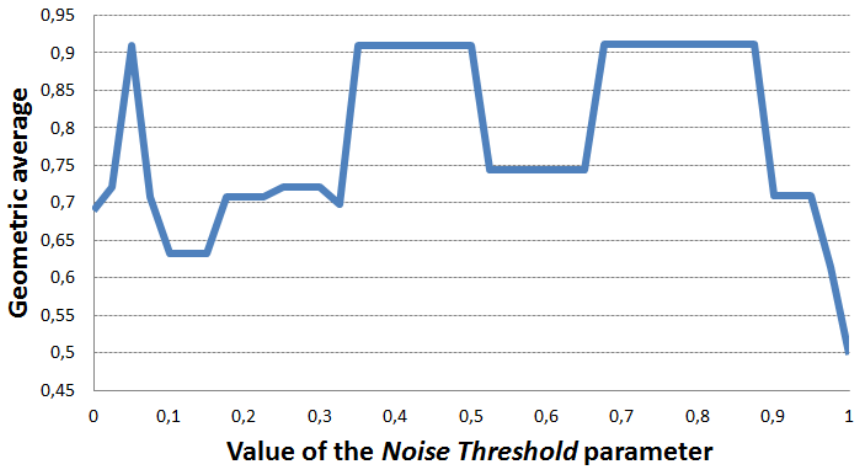


Figure 4.14: Parameter sensitivity analysis: Variation of the harmonic average of fitness and precision when varying the value of the *noise threshold* parameter.

By analyzing the graph, the models with higher harmonic average are pro-

duced when the parameter takes on a value between 0.675 and 0.875. The worst model is obtained when value 1 is assigned to the parameter.

4.4.3 Large-Scale Experiments

Empirical evaluation is often needed (and certainly recommended) when testing new process mining algorithms. In case of process mining, many experiments need to be conducted in order to prove that these algorithms or techniques can be applied in reality, and that the results are as expected. This is due to the richness of the domain. Process models can have a wide variety of routing behaviors, timing behavior, and second-order dynamics (e.g., concept drift). Event logs can be large or small and contain infrequent behavior (sometimes called noise) or not. Hence, this type of evaluation has to be conducted on a large scale. The execution and evaluation of such large-scale experiment results is a tedious and time-consuming task: it requires intensive human assistance by configuring each experiment's run and waiting for the results at the end of each run.

This can be greatly improved by using process mining workflows, as only one initial configuration is required. There are many examples for this use case within the process mining domain. Two applications of this use case are presented in Part III (i.e., Chapters 7 and 8), where we propose two frameworks that can be used to benchmark process discovery and concept drift techniques respectively through the use of large scale experiments.

4.4.4 Repeating Questions

Whereas the previous use cases are aimed at (data) scientists, process mining workflows can also be used to lower the threshold for process mining. After the process mining workflow has been created and tested, the same analysis can be repeated easily using different subsets of data and different time-periods. Without workflow support, this implies repeating the analysis steps manually or using *hardcoded* scripts that perform them over some input data. The use of scientific workflows is clearly beneficial: the same workflow can be replayed many times using different inputs where no further configuration is required.

There are many examples of this use case within the process-mining domain. An application of this use case in the domain of report generation over collections of data sets is described in Part IV (i.e., Chapter 10), where we analyzed the use of videolectures by thousands of students in hundreds of university

courses over the course of several years, where for each instance of a course, we provided lecturers with an automatically-generated report describing the videolecture-viewing behavior of students.

4.4.5 Interaction with Process Cubes

As discussed in Chapter 3, the cells of a *process cube* relate to set of events (and even to event logs). Such event data can be trivially used as input for a process mining workflow.

The idea is simple: A process mining workflow is designed for a specific purpose e.g., for creating a report of the performance of the process, including charts, tendencies, etc. Then, the same workflow can be executed many times using different cells of the cube as input.

The implementation of the process mining workflows and process cubes are related, but are independent from each other. The RapidMiner framework can be used as a library without a GUI. Therefore, the core of RapidMiner was integrated into the Process Mining Cube (PMC) tool described in Section 3.3. Within the cube, one can select cells and run a specific process mining workflow for them, in a similar fashion than visualizing the cell as an event log, or as a process model.

Figure 4.15 shows an example of a process mining workflow in RapidMiner using operators from the RapidProM extension. In this workflow, a process model is discovered from the event log provided as input, and the conformance checking of the event log with respect to the process model is performed. Finally, several result outputs of the conformance checking operator are written into files containing images, data or text.

Figure 4.16 shows an example of cells of the PMC, obtained from the Road Fines running example (see Example 2.4) where events are grouped based on the year in which they occurred. The cells corresponding to years 2000, 2004, 2008 and 2012 are selected, and they are used as input for the process mining workflow shown in Figure 4.15.

The concrete interaction between the process mining workflow and the process cube is performed through the following steps, shown in Figure 4.16:

1. Cells of the cube are selected.
2. A process mining workflow is selected.
3. The events in the cell can be merged and used once as input of the workflow, or the workflow can be executed for each cell separately.

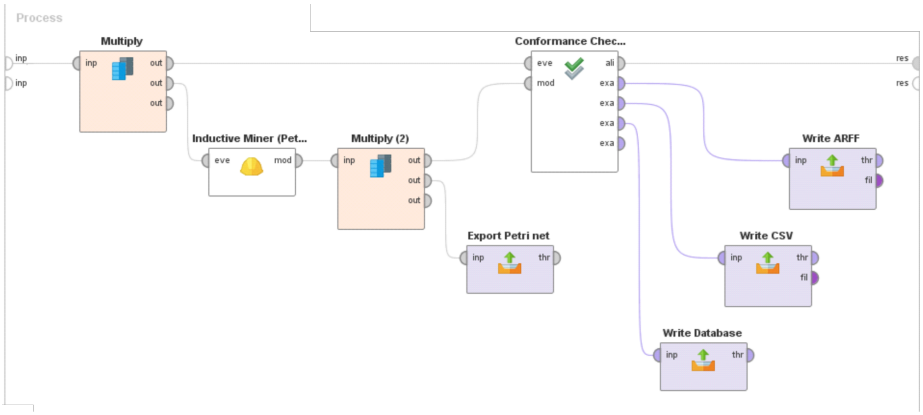


Figure 4.15: Process Mining Workflow implemented in RapidMiner using building blocks from the `RapidProM` extension: The input is an event log and the output is a conformance checking analysis of the event log (stored in several formats) and a model discovered from it.

Finally, the process mining workflow is executed.

The results produced by a process mining workflow are not visualized in the process cube (there are 200+ different types of result objects in *RapidMiner*, each one having several visualizers), and should be stored in disk and handled by the workflow itself using the available export operators in *RapidMiner* and RapidProM, using any naming scheme (e.g., "cell_X_model") that allows to store the results from different cells without overwriting them.

Note that the integration of the PMC and process mining workflows is not limited to only RapidProM operators. For example, one could use the events of a cell of the cube to perform clustering using standard data mining techniques.

4.5 Conclusions

This chapter presented a framework for supporting the design and execution of process mining workflows. As argued, scientific workflow systems are not tailored towards the analysis of processes based on models and logs. Tools like RapidMiner and KNIME can model analysis workflows but do not provide any process mining capabilities. The focus of these tools is mostly on traditional data mining and reporting capabilities that tend to use tabular data. Also more classi-

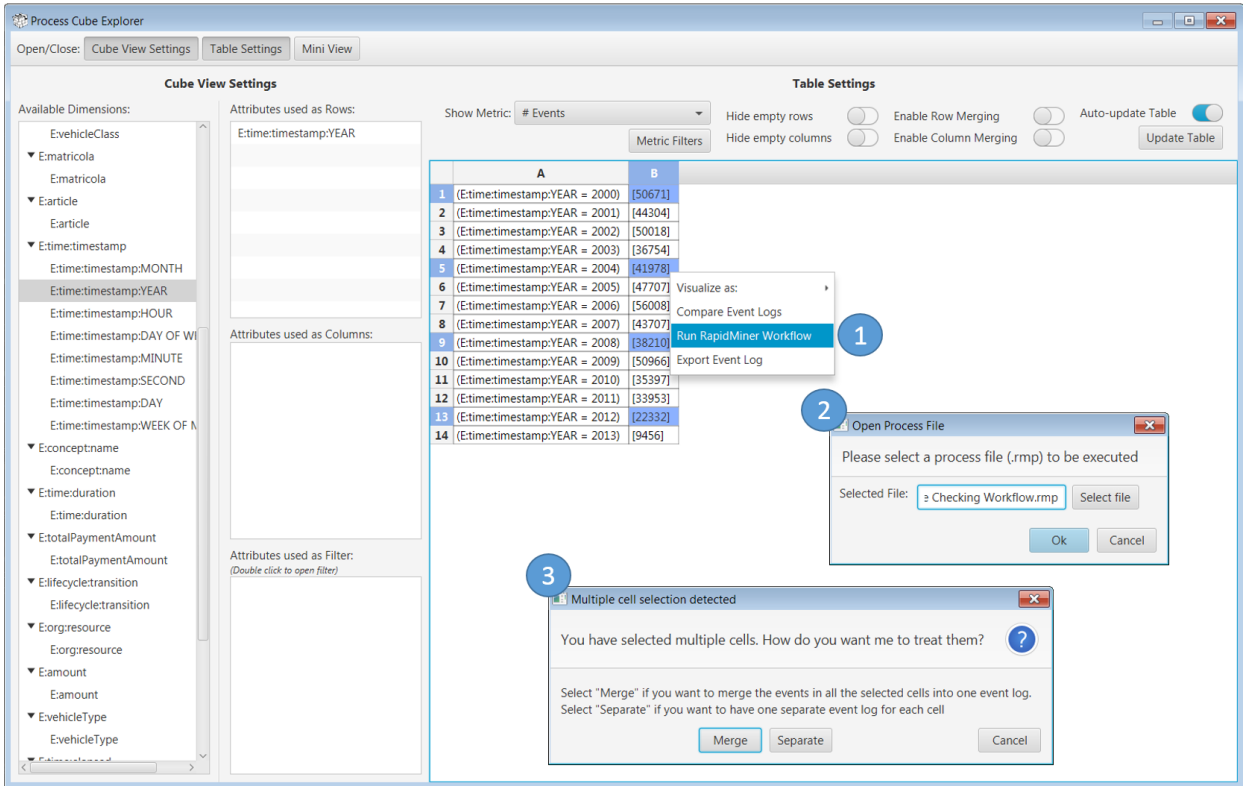


Figure 4.16: Selected cells of a Process Cube are used as input for the process mining workflow. The workflow can run for each cell independently, or for just once for the union of all the selected cells.

cal Scientific Workflow Management (SWFM) systems like Kepler and Taverna do not provide dedicated support for artifacts like process models and event logs. Process mining tools like ProM, Disco, Celonis, Everflow, QPR, MyInvenio, Minit, PM4PY, bupaR, etc. do not fully provide explicit workflow support.

We proposed generic *process mining building blocks* grouped into six categories. These can be chained together to create process mining workflows. We identified five generic use cases and provided conceptual and concrete workflows for these. The whole approach is implemented as RapidProM, which is a ProM-based extension for RapidMiner, an has been tested in several use cases. RapidProM is freely available at `http://www.rapidprom.org` and at RapidMiner Market place.

As a limitation, we would like to mention that the stability of RapidProM is not guaranteed with newer versions of the RapidMiner software. This will be caused by eventual changes in RapidMiner´s architecture over the years. In order to use RapidProM properly, please use the versions used here.

Chapter 5

Process Variant Comparison

Event logs (and process variants) can be compared to each other in order to find similarities and differences. Such similarities and differences can provide useful insights to business managers. Figure 5.1 positions the idea: one can compare a set of event logs in order to identify the strengths and weaknesses of each one, and their similarities and differences. Alternatively, one can also compare *process variants* obtained from the cells of a process cube (see Chapter 3).

Process similarity can be measured from several perspectives such as control-flow, performance, resource, and data. The combination of these process perspectives characterizes the *context* of a process [160]. The context of a process can be divided in four types:

Instance context refers to data attributes that can be related to individual process instances (e.g., type of customer, size of an order). This includes perspectives such as control-flow and performance.

Process context refers to data attributes that are not related to a single instance of the process, but to the process as a whole (e.g., workload and availability of resources).

Social context refers to data attributes that reflect the way in which people work together not within a specific process, but within a particular organization (e.g., team efficiency, relations between co-workers).

External context refers to data attributes that capture all other factors that are beyond the control sphere of an organization (e.g., weather, changing regulations).

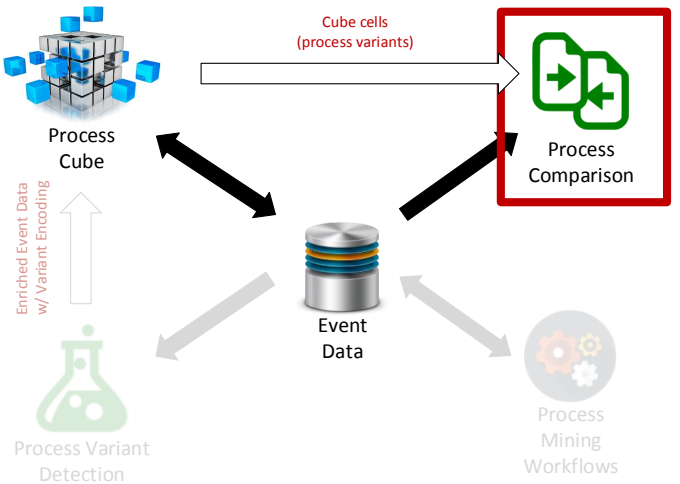


Figure 5.1: Overview of the scope of this chapter: event logs (e.g., process variants related to cells of a process cube) can be compared in order to identify their differences and similarities.

For the purpose of process comparison, we can split the context into two groups: behavioral and non-behavioral.

The *behavioral* context incorporates the perspectives that refer to "the way in which the process is executed" (i.e., the observed *behavior* of a process). This generally refers to the control-flow (i.e., *what* was executed) and performance (i.e., *when* was it executed) perspectives, which are a type of instance context. Behavior can be compared in order to find differences in the frequencies with which activities are executed, or in waiting and executing times. For example, one may notice that two process variants have slightly different ways of executing the same process. Furthermore, these variants may even have the same *control-flow*, but different *performance* for the same process.

The *non-behavioral* context incorporates the perspectives that refer to all the data attributes that do not describe behavior, but are related to it (e.g., the amount of the requested loan or the workload of people) These perspectives can have an influence on certain decisions made within the process, affecting its behavior. Sometimes, context data can be highly related to (and can even

determine) the way in which a process is executed. For example, the amount of a loan and the risk factor of a customer might determine if there will be extra financial checks or not before a loan is granted. In cases like this, such relations can be structured as *business rules*.

Similar behavior does not imply similar business rules (and vice versa). For example, two branches of a bank can reject the same percentage of loan applications using different thresholds to reject a loan. On the other hand, two branches can reject a different percentage of loan applications using the same threshold (and branches in richer regions would approve more loans).

This chapter presents a comprehensive technique to compare event logs (e.g., variants of a business process) in terms of behavior and business rules. The types of questions that our approach can answer are (not limited to):

- What are the differences in terms of behavior (e.g., frequency of occurrence, elapsed time) and business rules between two variants?
- In which points of the process (e.g., represented by a specific state or transition in a transition system) do these difference appear?
- Do the differences depend on certain behavioral and/or non-behavioral characteristics (e.g., *type* of a customer, workload of resources)?

The remainder of this chapter is structured as follows: Section 5.1 discusses related work. Section 5.2 details our technique for comparing two event logs in terms of their *behavior* and *business rules* and describes the result visualizations provided by our approach, including a discussion of the design principles behind the chosen design criteria. Section 5.3 describes the software tool that implements this approach. Section 5.4 illustrates the application of the tool in two experiments using synthetic and real data. Finally, Section 5.5 concludes the chapter.

5.1 Related Work

Substantial work has been done on comparing process variants. The corresponding papers can be grouped in three categories: model-based and log-based behavior comparison, and business rules comparison. The main difference between model-based and log-based behavior comparison approaches is that the first one requires process models as input and the second one requires event logs as input. Note that, model-based approaches could also be applied if the

model is not known, because models can be discovered from event logs and then used as input for the approach. However, the reliability of the results could be hampered by the fact that the structure of the models (hence, the detected differences) can be drastically affected by the choice of the discovery technique or its parameters. Finally, we include a discussion on stacking of decision trees and classifiers in general.

5.1.1 Model-based Behavior Comparison

Many model-based behavioral comparison [40, 73, 97, 99] and process model-matching [94, 95] techniques have been developed in recent years. La Rosa et al. [99] provide an overview of the different ways to compare and merge models. Most of them are based on control-flow comparison, where the structural properties of the models (represented as graphs) are compared (e.g., nodes and edges present in one of the models, but not in the other one).

A drawback of model-based approaches is that they are unable to detect differences in terms of frequency or other perspectives (e.g., elapsed time or costs). In other words, in model-based comparison the variants are compared in terms of their model structure whereas we aim to compare the behavior and business rules.

5.1.2 Log-based Behavior Comparison

One of the most recent approaches for log-based behavior comparison is by Van Beest et al. [151]. This technique is able to identify differences between two event logs by computing *frequency-enhanced prime event structures* (FPES) from the corresponding event logs, comparing the obtained FPES and report the results using two sets of textual statements: *control-flow* differences and *branching frequency* differences.

This approach has several advantages, such as the handling of concurrency in process behavior. However, the approach also has some limitations. First of all, the technique looks at the relative frequency, only. As such, when looking at branching frequency, it possibly returns a difference (if any), even though the branching point is actually reached very rarely. Also, no statistical significant tests are employed. Second, to determine branching points, they only look at the last activity independently of what activities were previously executed. As such - as we have verified by testing the reference implementation - it is unable to detect differences that refer to the activities preceding the last. Third, the approach considers event logs as sequences of event labels, thus ignoring all other

event attributes (e.g., timestamp, event payload). This limits the approach to detect only frequency differences. Differences in performance or other metrics cannot be obtained.

Another interesting process comparison technique was recently introduced in [193], where many process variants can be compared simultaneously. However, the focus is put on comparing only the performance of such process variants, while disregarding control-flow and other context differences.

Other approaches based on *sequence mining* such as [75, 101, 119, 144] tend to obtain over-fitting and complex rules (as indicated in [119] and [151]).

It is very gratifying for the authors to see that the work presented in this chapter was used to inspire further research as seen in [148], where they extended our work (published initially in [12] and later extended in [13]) by learning a directly-follows graph called *mutual fingerprint* from the event logs of two process variants. A mutual fingerprint is a lossless encoding of a set of traces and their duration using discrete wavelet transformation.

5.1.3 Business Rules Comparison

In machine learning, classifiers are usually compared in terms of their accuracy or predictive power: different types of classifiers can be trained, and the best are selected. This allows one to compare classifiers that are essentially different (e.g., decision trees and neural networks). However, accuracy and predictive power are not enough to understand business rules. Furthermore, black-box classifiers (e.g., neural networks) can indeed classify, but the underlying rules are not human-interpretable.

Since our approach compares business rules by comparing decision trees (see Section 5.2.2), we discuss decision tree comparison approaches as follows. The comparison of decision trees (and classifiers in general) is not process-specific, and it can be applied in many domains. Several authors have worked on the comparison of decision trees. [122] reports on an approach to compare branching conditions and to calculate a similarity score. However, such approaches tend to yield highly specific comparisons that have little relevance. For example, branch rules $x > 5$ and $x > 5.01$ are structurally different, but, in fact, the difference may be negligible from a domain viewpoint. In [121], a *similarity score* is obtained from the classification results, but such score does not point out the actual similarities and differences in decision trees (e.g., why or in which cases do differences occur) but it is only limited to somehow measure the degree of similarity.

In our approach, we use the classification predictions of existing decision trees to build a new decision tree that defines the conditions of agreement or disagreement between the predictions of the existing decision trees (see Section 5.2.2). This relates to *stacking* [192], a well-known concept of machine learning that consists of building a classifier based on the output of other classifiers. Unlike *bagging* and *boosting*, it is a different way of combining multiple classifiers, that introduces the concept of a *meta learner* that uses the predictions of other classifiers as input.

In machine learning, the whole ensemble or stack is commonly used as a single classifier. For example, in a 2-level stack an input instance is provided. Level-0 classifiers make a prediction, then level-1 classifiers take those predictions and make a new prediction, which is returned as the prediction (i.e., output) of the whole stack.

Unlike the above approach, we used the meta learner (e.g., a decision tree) for a different purpose: to understand under which conditions the classifiers (i.e., decision trees) predict the same class for a given instance, and under which conditions they make different predictions.

5.2 Process Variant Comparison

The process comparison approach presented in this chapter takes into account both behavior and business rules in order to perform a comprehensive analysis. For example, if a bank wants to reduce its number of branches in a city by merging them, it is probably best to merge those branches that are most similar to each other in terms of behavior and business rules, since this minimizes the efforts of the merge. If only behavior is compared, the resulting merged branch could have conflicting sets of business rules (e.g., different thresholds to give a loan) which could cause operational problems and ambiguity. For example, a loan application could be rejected if it is handled by an employee using a set of business rules (e.g., high thresholds), or accepted if it is handled by another employee using other rules (e.g., low thresholds). On the other hand, if only business rules are compared, the resulting merged branch could have conflicting control-flows, guided by employees that are used to execute different sets of tasks. For example, using the same loan approval thresholds, employees from one branch may do an additional check before approving a loan.

Figure 5.2 sketches the idea: two event logs (e.g., process variants obtained from a process cube) are compared for differences that are projected onto a transition system that represents the combined control-flow of both event logs,

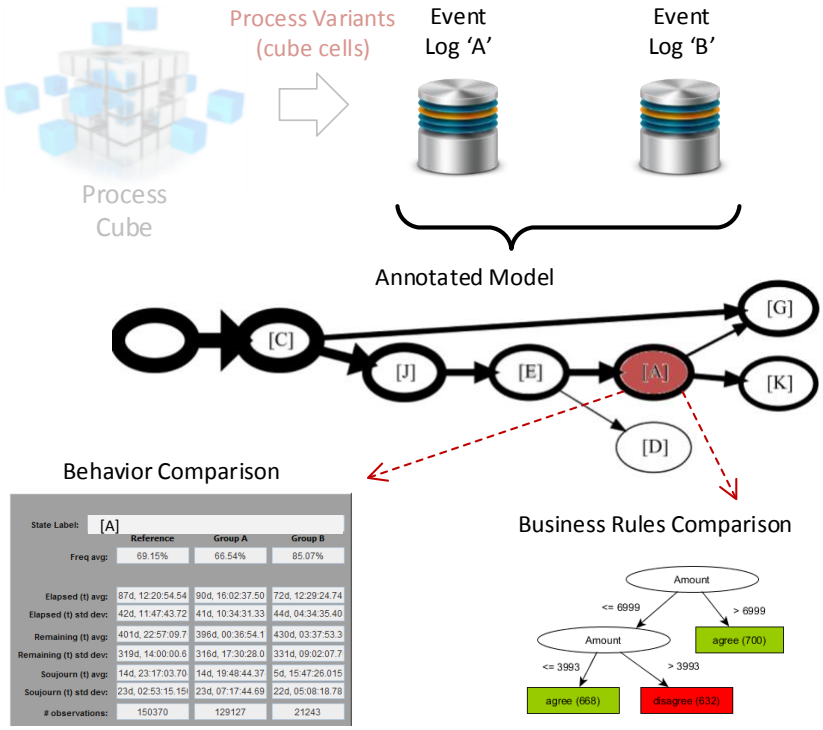


Figure 5.2: Overview of the approach: two event logs (e.g., cells from a process cube) are compared, producing a single annotated transition system that represents the combined behavior observed in both event logs, where the highlighted (i.e., colored) states and transitions highlight differences. Such states and transitions are interactive: when clicked, they show details of the actual differences. These can be related to behavior or business rules.

where states and transitions are colored to highlight differences in behavior or business rules. Then, highlighted states or transitions can be clicked on in order to show the details of such difference in terms of behavior or business rules.

The two event logs that are used for comparison can be extracted from the cells of a process cube (see Chapter 3). However, our technique is generic

enough to work with any event log i.e., not necessarily process variants. For example, the two event logs may have been extracted from different information systems (e.g., two branches of the same company or even two different companies). In the case that $k > 2$ event logs need to be compared, it is possible to create k pairs of 2 groups i.e., $k - 1$ groups are merged and compared with the remaining one ("one against the rest").

The remainder of this section describes our approach and how it can be used to compare behavior (Section 5.2.1) or business rules (Section 5.2.2) of different process variants.

5.2.1 Comparing Behavior

Event logs reflect the observed behavior of processes. Most process discovery techniques analyze this behavior from a pure control-flow perspective [156]. However, more behavioral related to other perspectives can be extracted from event logs (e.g., the cost of an activity, or the running time of a case). This information can be used to provide detailed insights about the behavioral differences between process variants.

In this section, we introduce a technique to extract and compare the behavior of event logs (e.g., process variants) using *annotations*: *measurements* are obtained from the event logs and are annotated on the states and transitions of a *transition system* that represents the combined behavior of both event logs. This technique is able to detect, for example, if one variant is significantly slower than the other, even if they have the same control-flow.

The remainder of this section is structured as follows. First, we provide a reminder of transition systems, and how they can be used to represent the combined behavior of two event logs. Second, we introduce the concept of measurements and how they can be annotated into a transition system. Then, we describe how, for each state and transition, the measurement values in the annotations of each variant are compared using statistical significance tests in order to detect relevant differences. Finally, we describe how the results are be visualized.

Recall: Transition Systems

This section consists of a brief recall of transition systems, as defined in Section 2.3.4, and can be skipped if the reader is familiar with these concepts.

Transition systems are directed graphs composed of *states* and of *transitions* between them. A transition is defined by an activity being executed, triggering

the current state of the process to move from a *source* state to a *target* state.

Prefixes of traces in an event log can be mapped to states and transitions using representation functions that define how these prefixes are interpreted.

The *state representation* function is defined as $r^s : \mathcal{E}^* \rightarrow \mathcal{R}^s$, where $\mathcal{R}^s$ is the universe of possible state representations. This function relates prefixes of traces to states in a transition system. Given an empty (prefix of a) trace, we denote the *empty state* as a special element $r^s(\langle\rangle) = \perp \in \mathcal{R}^s$.

The *activity representation* function is defined as $r^a : \mathcal{E} \rightarrow \mathcal{R}^a$, where $\mathcal{R}^a$ is the universe of all the possible activity representations. This function relates events to activities in a process.

The universe of all possible representations of transitions is defined as $\mathcal{R}^t \subseteq \mathcal{R}^s \times \mathcal{R}^a \times \mathcal{R}^s$. A transition $t \in \mathcal{R}^t$ is a triplet (s_1, a, s_2) where $s_1, s_2 \in \mathcal{R}^s$ are the source and target states and $a \in \mathcal{R}^a$ is the activity executed.

Given an event log L, a state representation function r^s and an activity representation function r^a, a *transition system* $TS^{(r^s, r^a, L)}$ is defined as a triplet (S, A, T) where $S \subset \mathcal{R}^s$ is the set of states, $A \subset \mathcal{R}^a$ is the set of activities and $T \subseteq (S \times A \times S)$ is the set of valid transitions between states (see Definition 2.14 in Section 2.3.4 for more details).

Given two event logs L_1 and L_2, where each event log is a set of traces, we can combine them into a new event log $L' = L_1 \cup L_2$, which is the union of both sets of traces. Therefore, given the two event logs L_1 and L_2, a state representation function r^s and an activity representation function r^a, we can define $TS^{(r^s, r^a, L_1 \cup L_2)}$ as the transition system that represents the combined behavior of both event logs L_1 and L_2.

Measurements and Annotations

In order to compare event logs, we need to introduce the *measurements* that we will use for comparison. Measurement functions are computed as functions of event attributes contained in the events of a trace.

Definition 5.1 (State Measurement Function)**.** *Given an event log L and a transition system $TS^{(r^s, r^a, L)} = (S, A, T)$ with a state representation function r^s and an activity representation function r^a, a state measurement function, defined as $sm_{r^s} : \mathcal{E}^* \times \mathcal{R}^s \rightarrow \mathbb{B}(\mathbb{R})$, is a function that relates traces $\sigma \in \mathcal{E}^*$ and states $s \in \mathcal{R}^s$ to a multiset of numerical measurements.*

Note that for a given trace and state, multiple measurement values can be obtained, e.g., if a state is visited twice within the same trace (e.g., in a loop in the process), it can lead to two different *elapsed time* values.

For example, it is possible to measure whether or not a certain state s in a state representation r^s is reached during the process' execution recorded in a trace σ:

$$sm^{occur}_{r^s}(\sigma, s) = \begin{cases} [1] \text{ if } \exists \sigma' \in pref^{\diamond}(\sigma) : r^s(\sigma') = s \\ [0] \text{ otherwise} \end{cases} \tag{5.1}$$

It is also possible to measure the *elapsed time* between the beginning of a trace σ and the visit of a state s using a state representation r^s:

$$sm^{elapsed}_{r^s}(\sigma, s) = \biguplus_{\substack{\sigma' \in pref^{\diamond}(\sigma), \sigma' \neq \langle\rangle \\ r^s(\sigma') = s}} [\#_{time}(\sigma'(|\sigma'|)) - \#_{time}(\sigma'(1))] \tag{5.2}$$

Note that in this case, the same trace can contain more than one prefix that reach the same state, leading to multiple measurement values. Also note that we use the notation $\uplus$ to indicate the sum of multisets, while we use the notation $\cup$ to indicate the union of sets (see Section 2.1).

Definition 5.2 (Transition Measurement Function)**.** *Given a state representation function r^s and an activity representation r^a, a transition measurement function $tm_{(r^s, r^a)} : \mathcal{E}^* \times \mathcal{R}^t \rightarrow \mathbb{B}(\mathbb{R})$, is a function that relates a trace $\sigma \in \mathcal{E}^*$ and a transition $t \in \mathcal{R}^t$ to a multiset of numerical measurements.*

For example, it is possible to measure whether a certain transition t is executed in a given trace σ:

$$tm^{occur}_{(r^s, r^a)}(\sigma, t) = \begin{cases} [1] \text{ if } \exists_{\sigma' \in pref^{\diamond}(\sigma), \sigma' \neq \langle\rangle} \left(r^s(pref^{|\sigma'|-1}(\sigma')), r^a(\sigma'(|\sigma'|)), r^s(\sigma')\right) = t \\ [0] \text{ otherwise} \end{cases} \tag{5.3}$$

It is also possible to measure the *elapsed time* of a trace until a transition is triggered within the trace:

$$tm^{elapsed}_{(r^s, r^a)}(\sigma, t) = \biguplus_{\substack{\sigma' \in pref^{\diamond}(\sigma), \sigma' \neq \langle\rangle \\ \left(r^s(pref^{|\sigma'|-1}(\sigma')), r^a(\sigma'(|\sigma'|)), r^s(\sigma')\right) = t}} [\#_{time}(\sigma'(|\sigma'|)) - \#_{time}(\sigma'(1)] \tag{5.4}$$

States and transitions can be *annotated* with the measurements obtained from an event log [163].

Definition 5.3 (Annotation Function). *Given a state measurement function sm, a transition measurement function tm and an event log L, an annotation function $an^{(sm,tm,L)} : (\mathcal{R}^s \cup \mathcal{R}^t) \to \mathbb{B}(\mathbb{R})$, is a function that, given a state $s \in \mathcal{R}^s$ or transition $t \in \mathcal{R}^t$, produces a multiset of numerical measurements. The annotation function is defined as:*

$$an^{(sm,tm,L)}(x) = \begin{cases} \biguplus_{\sigma \in L} sm(\sigma, x) \text{ if } x \in \mathcal{R}^s \\ \biguplus_{\sigma \in L} tm(\sigma, x) \text{ if } x \in \mathcal{R}^t \end{cases}$$

Creating Annotated Transition Systems from Event Logs

In order to compare process variants, we need to be able to compare the annotations that are produced for each state and transition of a transition system (see Def. 2.14). Hence, we introduce *annotated transition systems* which allows one to annotate a transition system with multiple annotation functions.

Definition 5.4 (Annotated Transition System). *Given two event logs L_1 and L_2, state and activity representation functions r^s and r^a, state and transition measurement functions sm and tm, we define an annotated transition system $ATS^{(r^s,r^a,L_1,L_2,sm,tm)}$ as the triplet $(TS^{(r^s,r^a,L_1 \cup L_2)}, an^{(sm_{r^s},tm_{(r^s,r^a)},L_1)}, an^{(sm_{r^s},tm_{(r^s,r^a)},L_2)})$ consisting of a transition system $TS^{(r^s,r^a,L_1 \cup L_2)}$ and two annotation functions $an^{(sm_{r^s},tm_{(r^s,r^a)},L_1)}$ and $an^{(sm_{r^s},tm_{(r^s,r^a)},L_2)}$.*

Note that the transition system $TS^{(r^s,r^a,L_1 \cup L_2)}$ uses all the traces contained in the event logs L_1 and L_2. All the annotations in an_1 are related to the event log L_1 and all the annotations in an_2 are related to the event log L_2. This allows us to extract only the annotations related to a specific event log for any given state or transition.

Figure 5.3 shows an example of an annotated transition system. It is the result of annotating a simplified transition system related to the journal revision process using the event log shown in Figure 1.4 which is split into two sublogs: L_1 contains the first two traces (i.e., no extra reviews needed) and L_2 contains the third trace (i.e., extra review needed), the state representation function $r^s(\sigma) = \{\#_{activity}(e) | e \in \sigma\}$ for $\sigma \in P_L$, the activity representation function $r^a(e) = \#_{activity}(e)$ for $e \in E_L$, the state measurement function sm_{r^s} defined in Equation 5.1 and the transition measurement function $tm_{(r^s,r^a)}$ defined in Equation 5.3. Note that the measurements corresponding to all three traces are included in the annotations.

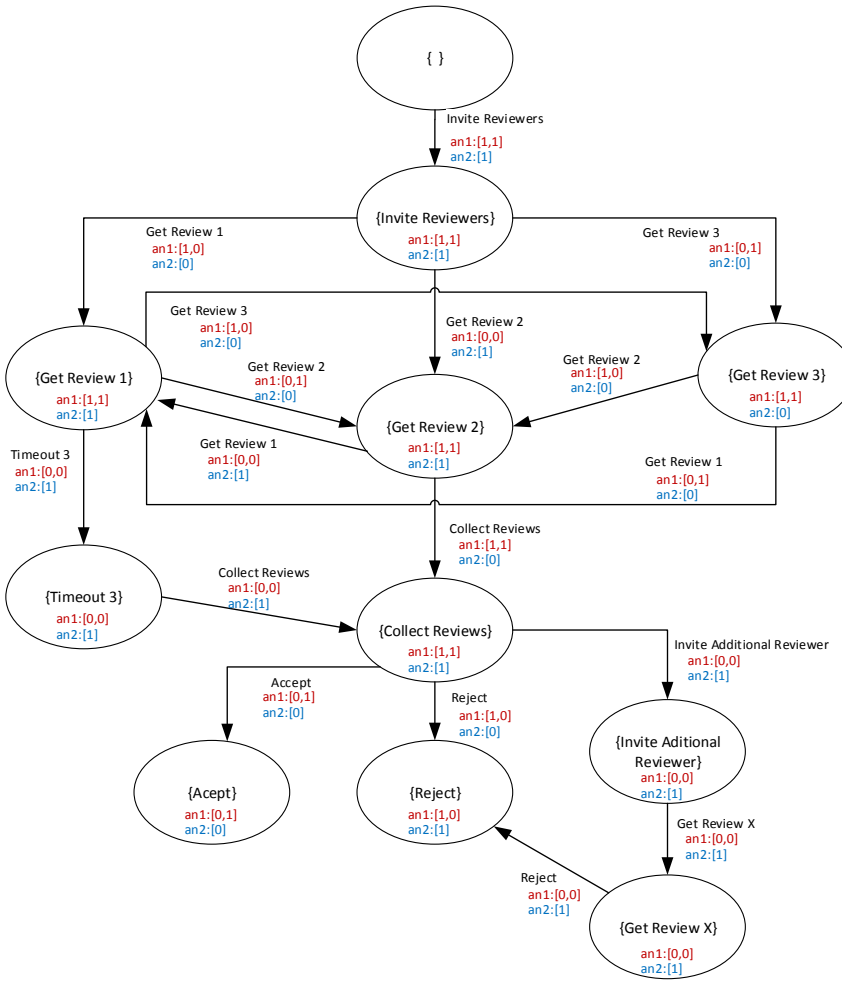


Figure 5.3: A simplified version of the transition system presented in Figure 2.4 is annotated with the annotation functions an_1 and an_2 with *occurrence* state and transition measurement functions defined in Equations 5.1 and 5.3. Annotations are represented as text under the node and edge labels. Blue-colored annotations correspond to an_1 and red-colored annotations correspond to an_2.

State and Transition Comparison using Annotations.

The annotations in states and transitions can be compared in order to detect differences between them.

Definition 5.5 (Comparison Oracle)**.** *Given two multisets of numerical measurements (i.e., annotations), the comparison of annotations can be abstracted as a comparison oracle that is defined as the function* $diff : \mathbb{B}(\mathbb{R}) \times \mathbb{B}(\mathbb{R}) \rightarrow Bool$*, that decides whether there are significant differences between such multisets (i.e., true) or not (i.e., false).*

This comparison oracle is unrelated to the notion of cases and processes, as it works over numerical annotations. Such annotations incorporate all the process-specific information of the event data.

Given an $ATS = \big((S, A, T), an_1, an_2\big)$, for each element $x \in S \cup T$ we want to detect differences by evaluating $diff\,(an_1(x), an_2(x))$.

The framework is irrespective of the employed comparison oracle. However, in this chapter we propose two concrete oracles. If the annotation values follow a normal distribution, we use the two-tailed Welch's T-test, also known as the "two-tailed T-test with different variances" [190]. Otherwise, we use the non-parametric rank-based Mann-Whitney U-test [113]. This test is the non-parametric version of a T-test (i.e., it does not assume normality).

Statistical significance tests only indicate if there is a difference big enough to be significant, but do not take into account the magnitude of the actual difference. For example, a difference of one second or one month can be statistically significant, depending on the sparsity or skewness of the annotations. To address this, the *effect size* is measured in order to indicate the magnitude of the difference. Note that the effect size is only calculated when a statistically significant difference is detected. This means that states and transitions that do not contain statistically-significant differences are not measured for effect size.

Definition 5.6 (Effect Size Oracle)**.** *The effect size oracle is defined as the function* $eff : \mathbb{B}(\mathbb{R}) \times \mathbb{B}(\mathbb{R}) \rightarrow \mathbb{R}$*, which given two multisets of measurements, returns the size of the effect (i.e., how small or large is the difference) and the sign of the difference (+/-) in a certain scale.*

In this chapter, we use Cohen's d [38] to measure effect size, which measures the difference of sample means in terms of pooled standard deviation units. Cohen relates ranges of d values to effect size categories: $d = \pm 0.2$ is considered as a *small effect*, $d = \pm 0.5$ is considered as a *medium effect* and $d = \pm 0.8$ is considered as a *large effect*. However, other effect size measurements could be used instead.

Visualization of Results

This section discusses a way to visualize the behavior comparison results described above. Design principles of transition systems including thickness, color, shape, and layout can be used to represent information. The design principles that we used in this chapter are:

- Processes can be visually represented as transition systems (see [162]).
- Thickness of elements in a graph can be used to represent numerical properties (e.g., existing process mining tools use this to show delays and frequencies).
- Categories (e.g., related to Cohen's d) can be represented by color scales (see [22]).

The most common design principles used in the process mining software industry (e.g., tools like Disco[1], Celonis[2], Minit[3], MyInvenio[4], Perceptive Process Mining[5], SNP Business Process Analysis[6], and many more) are the use of *thickness* to represent scalar values (e.g., frequency of occurrence) and the use of *color* to highlight elements of the model (e.g., nodes, arcs). In this chapter, we do not claim a contribution in terms of visualization, and we adhere to the common industry design principles mentioned above.

Thickness is especially useful to represent numerical attributes due to the high number of different values it can take (i.e., a small range of real numbers) and by the fact that it is unidimensional and scalar (i.e., when comparing two thicknesses, one can quickly identify the magnitude of their difference). For example, if a line is two times thicker than another line, then its numerical attribute value is two times bigger.

In this chapter, we use thickness to represent behavioral properties of the process (e.g., frequency, elapsed time). The behavioral property used to calculate thickness is configured by the user: Given an annotated transition system $ATS = \big((S, A, T), an_1, an_2\big)$, for each element $x \in S \cup T$, the *thickness* of the corresponding node (if $x \in S$) or arc (if $x \in T$) is proportional to the mean value of $an_1(x) \uplus an_2(x)$, i.e., the average value of the annotations associated with x

[1] https://fluxicon.com/disco
[2] www.celonis.com/en
[3] www.minitlabs.com
[4] www.my-invenio.com
[5] www.lexmark.com
[6] www.snp-bpa.com

and computed on the merged log. Note that the thickness property can provide insights about the overall behavior of both variants combined. However, other alternatives representations are possible, e.g., the average difference of $an_1(x)$ and $an_1(x)$ can be used instead in order to highlight the states or transitions with biggest absolute differences in their annotation values.

Figure 5.4 shows an example of this visualization using the exact same ATS

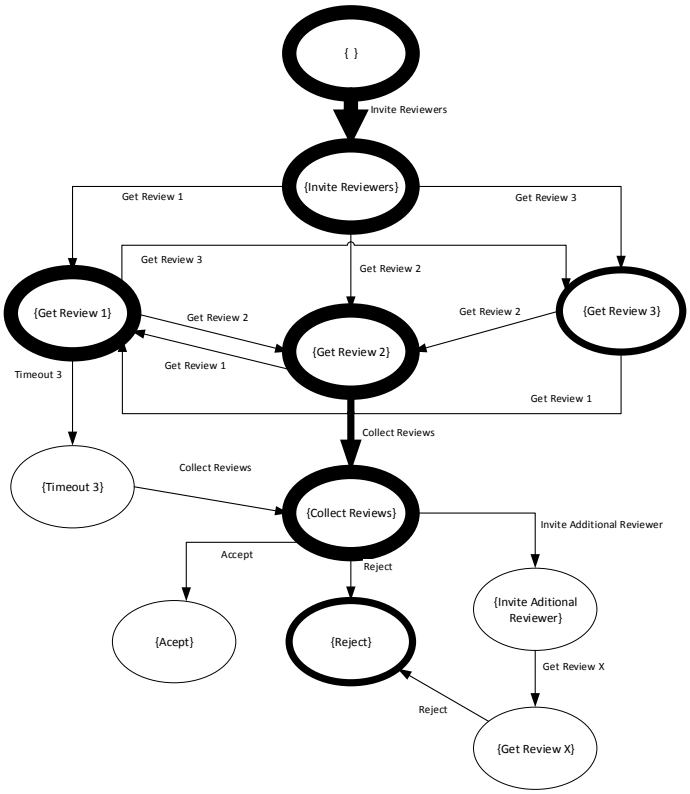


Figure 5.4: An example of how the annotations are translated to the thickness of the transition's arcs and state's node borders using the annotated transition system shown in Figure 5.3. In this case, thickness represents the combined frequency of occurrence.

presented in Figure 5.3, which uses the state representation function $r^s(\sigma) = \{\#_{activity}(e)|e \in \sigma\}$ for $\sigma \in P_L$, the activity representation function $r^a(e) = \#_{activity}(e)$ for $e \in E_L$, the state measurement function sm_{r^s} defined in Equation 5.1 and the transition measurement function $tm_{(r^s,r^a)}$ defined in Equation 5.3. In this case, the annotations are represented as thickness instead of text. Note that in this figure, thickness represents the frequency of occurrence.

Color is commonly used to highlight elements, because the human eye is very susceptible to color contrast. This fact can be used to draw the attention of the user to the parts of the process that contain differences.

Given an annotated transition system $ATS = \big((S, A, T), an_1, an_2\big)$, for each element $x \in S \cup T$, the corresponding node (if $x \in S$) or arc (if $x \in T$) will be colored black or white (depending whether it is a transition or a state, respectively) if there are *no statistically-significant differences* detected in that element. On the other hand, if a statistically-significant difference is detected, a different color will be used.

In this latter case, the specific color used depends on the type and magnitude of the difference found: For differences in terms of behavior, the color used will depend on the *effect size* of the difference (defined in Section 5.2.1). Please note that our approach can analyze only one annotation (e.g., frequency, elapsed time) at a time, and it is defined by the user. The selected annotation is then used for all the statistical tests and effect size calculations.

Different colors are used according to Cohen's d ranges of effect size values (8 value ranges in total: four for positive d's and four for negative d's). Colors with higher intensity (i.e., darker) represent larger effect sizes (i.e., more relevant differences), whereas colors with low intensity (i.e., lighter) represent smaller effect sizes (i.e., less relevant differences).

Blue-based colors mean that the average value of the annotation of a state or transition in a first event log is higher than in a second event log and red-based colors mean the opposite. Figure 5.5 shows an example of a transition system in which some states and transitions are colored blue or red if they present statistically significant differences in their annotations.

The color scheme (blue and red) was obtained from `http://colorbrewer2.org` (introduced in [22]) from the selection of *diverging* color schemes (i.e., where white is the *median* color and the two extremes are different non-white colors) with eight levels that were also identifiable by color-blind users. In [22], Brewer discusses how these schemes are generated and how they can be used to represent categories.

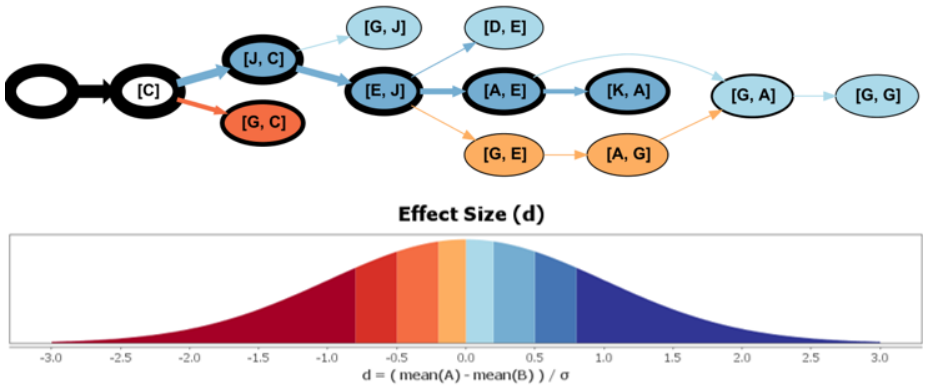


Figure 5.5: Example of an annotated transition system colored with the results of statistical significance tests and effect size oracle. States and transitions that do not contain statistically significant differences (hence, the effect size is not measured) are colored white and black respectively.

5.2.2 Comparing Business Rules

When a state of a transition system has more than one outgoing transition, a *decision* needs to be made: only one of the outgoing transitions can be executed. Such states are called as *decision points*.

Definition 5.7 (Decision Point). *Given a transition system $TS = (S, A, T)$, a state $s \in S$ is a* decision point *if it has at least two different outgoing transitions, namely* $\exists a', a'' \in A, \exists s', s'' \in S : (p, a', s') \in T \wedge (p, a'', s'') \in T \wedge a' \neq a''$.

As discussed earlier, the non-behavioral context of a process can be directly related to its observed behavior. Such context information is present in event data in the form of data attributes (e.g., workload, type of customer). For any decision point, the context information (contained in the event log in the form of data attributes) can be used to discover business rules that capture the data conditions that lead to a control-flow decision (e.g., if the amount of a requested loan is higher than X, then the loan is rejected). Similarly to what was proposed in [44], we use *decision trees* [124] to discover data conditions (i.e., business rules) at the decision points.

Given a transition system, for each decision point, the business rules comparison is performed as follows:

1. Extract *observation instances* from the event logs of the two variants. In abstract terms, an observation instance relates, for a given trace prefix in an event log, the execution of one of the outgoing transition of the decision point (i.e., target or *class* variable) to a set of attribute values (i.e., descriptive variables)
2. For each variant, a *decision tree* is built using the respective observation instances.
3. *Compare* the decision trees for differences.

Note that in step 1, all the necessary process-specific information contained in the event data is embedded into the observation instances. Therefore, steps 2 and 3 are not process-specific. Given a collection of observation instances (i.e., value vectors), any classification technique can be used instead. Step 3 proposes a novel way to compare decision trees according to whether they agree or disagree in their predictions. These three steps are detailed in the remainder of this section.

Creation of the Observation Instances

Since we are interested in analyzing how data attributes can explain the control-flow decisions of a process variant (i.e., business rule), for each decision point we need to create observation instances that contain such data attributes. These instances will be used to train and test decision trees.

Let L be an event log and $TS^{(r^s,r^a,L)} = (S, A, T)$ be a transition system. Let $DP \subseteq S$ be the set of decision points of the transition system TS. For each decision point $d \in DP$, an *observation instance* associated to d is a pair associating a set of attribute values to the next activity $a \in A$ being executing, as observed in the event log. For the remainder of this section, we introduce function:

$$values : \mathcal{E}^* \rightarrow (\mathcal{N} \not\rightarrow \mathcal{V})$$

that, for each trace (prefix) σ, returns a function $f = values(\sigma)$ that relates attributes $n \in \mathcal{N}$ to values $f(n) \in \mathcal{V}$. For each attribute n in the domain of f, $f(n)$ returns the latest value that n has taken on in trace (prefix) σ.

With this helper function at hand, we can precisely define the set of observation instances associated with an event log.

Definition 5.8 (Observation instances)**.** *Let L be an event log and let P_L be the set of all prefixes of all traces in L. Let $TS^{(r^s,r^a,L)} = (S, A, T)$ be a transition*

system. Given a decision point $d \in DP \subseteq S$, the multiset of observation instances $I_d^{(L,r^s,r^a)} \in \mathbb{B}\big((\mathcal{N} \not\rightarrow \mathcal{V}) \times A\big)$ related to d is defined as:

$$I_d^{(L,r^s,r^a)} = \biguplus_{\substack{\sigma \in P_L \setminus \{\langle\rangle\}, \\ r^s(pref^{|\sigma|-1}(\sigma)) = d}} [(values(pref^{|\sigma|-1}(\sigma)), r^a(\sigma(|\sigma|)))]$$

where $r^a(\sigma(|\sigma|)) \in A$ is the class attribute, namely the activity that is observed being executed (i.e., the last activity executed in the trace σ).

Table 5.1 shows an example in the context of the journal revision process, where the observation instances were obtained using the event log shown in Figure 1.4 and the transition system for this event log shown in Figure 2.4 for the decision point given by the state "Invite Reviewers". Note that these

Table 5.1: Fragment of a set of Observation Instances related to the journal revision process in the decision point given by the state "Invite Reviewers".

Trace ID	Observation Instance
3025	({f(activity) = "Invite Reviewers", f(timestamp) = 01-08-2017}, "Get Review 1")
3026	({f(activity) = "Invite Reviewers", f(timestamp) = 14-08-2017}, "Get Review 3")
3027	({f(activity) = "Invite Reviewers", f(timestamp) = 06-09-2017}, "Get Review 2")

observation instances have different target activities, i.e., "Get Review 1", "Get Review 2", and "Get Review 3". Also note that if any of the events "Invite Reviewers", "Get Review 1", "Get Review 2", or "Get Review 3" had an extra data attribute (e.g., $resource = John$), its latest value would also be included in the observation instance, even if the other events do not contain such attribute.

Training and Testing Decision Trees to Obtain Business Rules

Once a multiset of observation instances I_d has been created for each decision point d, *decision trees* are trained with such instances. Concretely, we use the C4.5 algorithm to train decision trees [125]. Even though other decision-tree training algorithms can be employed, we opted for C4.5 because of its ability of dealing with noise and missing values. The decision tree trained with the set of observation instances I_d is denoted as DT_d.

Once a decision tree has been trained, it can be used to represent the business rules of a process variant at a particular decision point. The decision tree can also be used to predict the next activity that is going to be executed when

a certain decision point is reached based on attribute values. For later usage, assume the function that describes the decision tree semantics:

$$eval_{DT_d} : (\mathcal{N} \nrightarrow \mathcal{V}) \rightarrow A$$

Given a function f that assigns values to certain attributes, $eval_{DT_d}(f)$ predicts the next activity $a \in A$ to be executed when reaching a certain decision point d when attributes take on values according to function f. Note that the attributes that are given values in the set of observation instances might be different from those used by decision trees to classify new instances. In these cases, the additional attributes are discarded and the missing attributes are set as *missing*. The C4.5 algorithm can deal with missing values by assuming that they can take any value.

Comparing Decision Trees to Identify Differences in Business Rules

Let L_1 and L_2 be two process variants and $TS^{(r^s, r^a, L_1 \cup L_2)} = (S, A, T)$ be a transition system. Given a decision point $d \in S$, we can compare the decision trees DT_d^1 and DT_d^2 respectively trained with instances $I_d^{(L_1, r^s, r^a)}$ and $I_d^{(L_2, r^s, r^a)}$ by performing a *cross-validation*. Figure 5.6 illustrates the idea. For each ob-

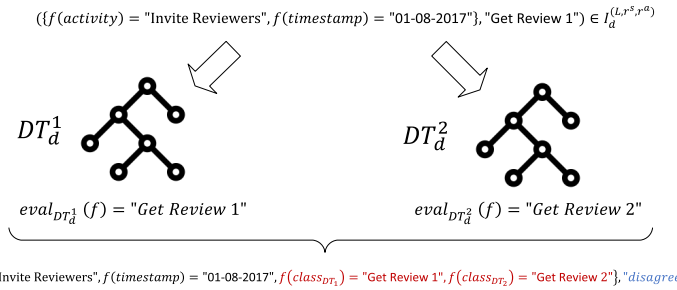


Figure 5.6: Example of a decision tree comparison using the first observation instance shown in Table 5.1. An observation instance is evaluated by both decision trees DT_d^1 and DT_d^2. In this case, they classify the instance differently. An *extended observation instance* is created from the observation instance by adding the classification of both trees as attributes (highlighted in red), and changing the target variable to “disagree” (highlighted in blue). If the trees would have predicted the same class, the target variable would be “agree”.

servation instance in $I_d^{(L_1 \cup L_2, r^s, r^a)}$, we classify it using the decision trees DT_d^1 and DT_d^2. The more instances $(f, a) \in I_d^{(L_1 \cup L_2, r^s, r^a)}$ have the same prediction by both trees, the more similar are DT_d^1 and DT_d^2 and, hence, the fewer differences exist in business rules between the two variants for the decision point d. In this thesis, we define the degree of similarity as simply the percentage of observation instances that yield the same prediction in both trees. Formally, this is defined as follows.

Definition 5.9 (Degree of Similarity of Decision Trees)**.** *Given two decision trees DT_d^1 and DT_d^2 and a set of observation instances I_d, the degree of similarity of DT_d^1 and DT_d^2 can be measured as:*

$$similarity(I_d, DT_d^1, DT_d^2) = \frac{|\{(f,a) \in I_d | eval_{DT_d^1}(f) = eval_{DT_d^2}(f)\}|}{|I_d|}$$

Cross-validation results relate to the degree of exchangeability of business rules between process variants. It provides answers to the following question: "What if a process variant is executed using the business rules of another variant?" If the degree of similarity is 1 (i.e., perfectly similar), then the business rules of the variants are exchangeable, and the observed behavior after exchanging the rules should not be affected. On the other hand, if the degree of similarity is is low, this means that the business rules of the variants are different, and the observed behavior should be different as well.

Cross-validation is useful to quantify the degree of exchangeability (thus, similarity) of business rules between variants, but does not provide any information about on which specific situations they agree or disagree. To address this, we analyze under which conditions the decision trees of two variants *agree* (i.e., classify an observation instance in the same class) or *disagree*. The disagreement conditions are relevant because they indicate variants business rules. Furthermore, they describe conflicts in terms of business rules. For example, if the threshold to accept a loan request is 1000 in one variant, and 2000 in another variant, this means that both variants accept loans up to 1000, both variants reject loans of 2000 or more, and there there is a disagreement between them in the amount range $]1000, 2000[$. This means that the business rules of the variants predict different control-flow paths (i.e., accept or reject the loan) for any loan having an amount in that range.

For this purpose, we *stack* decision trees: we build a new decision tree using *extended observation instances*, which are observation instances extended with the classification predictions of the decision trees of the two variants, as shown in Figure 5.6. Formally, they are defined as follows.

Definition 5.10 (Extended Observation Instances)**.** *Given an event log L, a transition system $TS^{(r^s,r^a,L)} = (S, A, T)$, for each decision point $d \in S$, we create a multiset of extended observation instances (denoted as $EI_d^{(L,r^s,r^a)}$) by extending all the instances $(f, a) \in I_d^{(L,r^s,r^a)}$ with the classification results of the two decision trees DT_d^1 and DT_d^2, and whether they agree or disagree in their classification, regardless of the actual value of a:*

$$EI_d^{(L,r^s,r^a)} = \biguplus_{\substack{(f,a)\in I_d^{(L,r^s,r^a)}:\\ eval_{DT_d^1}(f)=eval_{DT_d^2}(f)}} [(f', agree)] \quad \biguplus_{\substack{(f,a)\in I_d^{(L,r^s,r^a)}:\\ eval_{DT_d^1}(f)\neq eval_{DT_d^2}(f)}} [(f', disagree)]$$

where f' is used as a replacement of $f \oplus$ ("$class_{DT_1}$", $eval_{DT_d^1}(f)) \oplus$ ("$class_{DT_2}$", $eval_{DT_d^2}(f)$).[7]

Function f' contains the attribute value assignments of f and, additionally, the evaluation results of f using both decision trees (attributes $class_{DT_1}$ and $class_{DT_2}$ respectively).

The extended observation instances EI_d are used to train a new decision tree EDT_d, which is used to correlate the (dis)agreement of the decision trees of two variants with the attribute values of one of the trees. Note that EDT_d simply classifies the extended observation instances EI_d into two classes: "agree" and "disagree" using all the other available attributes. All the evaluations mentioned above for a decision point are summarized into a *decision point matrix*, as illustrated in Figure 5.7. Each cell in this matrix (i.e., intersection of a row and a

	DT_d^1	DT_d^2	EDT_d
$I_d^{(L_1,r^s,r^a)}$	DT_d^1 tested with $I_d^{(L_1,r^s,r^a)}$	DT_d^2 tested with $I_d^{(L_1,r^s,r^a)}$	EDT_d tested with $EI_d^{(L_1,r^s,r^a)}$
$I_d^{(L_2,r^s,r^a)}$	DT_d^1 tested with $I_d^{(L_2,r^s,r^a)}$	DT_d^2 tested with $I_d^{(L_2,r^s,r^a)}$	EDT_d tested with $EI_d^{(L_2,r^s,r^a)}$
$I_d^{(L,r^s,r^a)}$	DT_d^1 tested with $I_d^{(L,r^s,r^a)}$	DT_d^2 tested with $I_d^{(L,r^s,r^a)}$	EDT_d tested with $EI_d^{(L,r^s,r^a)}$

Figure 5.7: Abstract representation of a Decision Point Matrix for a decision point d, given a transition system $TS^{(r^s,r^a,L)}$ and two process variants L_1 and L_2 where $L = L_1 \cup L_2$. Each row header corresponds to a multiset of observation instances. Each column header corresponds to a decision tree. Each cell (i.e., intersection of a row and a column) contains the classification results of a multiset of observation instances (row) using a decision tree (column).

column) corresponds to the *classification results* of a multiset of observation instances (indicated by the row) using a decision tree (indicated by the column).

[7]The operator $\oplus$ is used to override the definition of a function $f' = f \oplus (a, b)$ with $f'(a) = b$ and $f'(x) = f(x)$ for $x \neq a$.

The way in which the classification results are presented to the user is described in Section 5.2.2.

In transition systems with many decision points, it is slow and cumbersome to explore all of them in order to find differences in the business rules. To address this issue, we want to identify the decision points that have the highest differences.

We can measure the differences between variants in a decision point according to an *agreement score*, defined as the percentage of extended observation instances that were classified as "agree" by the new decision tree EDT_d (i.e., the decision trees DT_d^1 and DT_d^2 agreed on their classification). Low agreement scores indicate that there are more differences in the decision trees, thus, in the business rules of the process variants.

Visualization of Results

This section describes how the business-rules comparison results can be visualized.

To point out the decision points with larger differences, users can indicate a threshold for agreement scores. For each decision point, if the agreement score is below the threshold, then the element has a considerable difference and is colored bright-red, otherwise is colored light-gray. We chose these two colors because of their contrast with black and white: bright-red is highly visible when contrasted with black and white, and light-gray is not as contrasting, but enough to inform that there is a small difference.

Figure 5.8 shows an example of a transition system in which decision points are highlighted order to show where the differences are.[8]

If the user clicks in a highlighted decision point (where the differences score lower than the agreement threshold described above), a decision point matrix is shown (see Figure 5.7).

The classification results contained in each cell of a decision point matrix may contain many insights about how the decision tree classifies observation instances, and they may be compared to other cells in order to detect differences. Concretely, we use two ways to present classification results, as shown in Figure 5.9.

The first one is to use a pie chart that shows the percentage of correctly and incorrectly classified instances. It is also used for representing the percentage of

[8]Note that the example transition system used in this figure differs from previous examples. It has been added to better illustrate the visualizations.

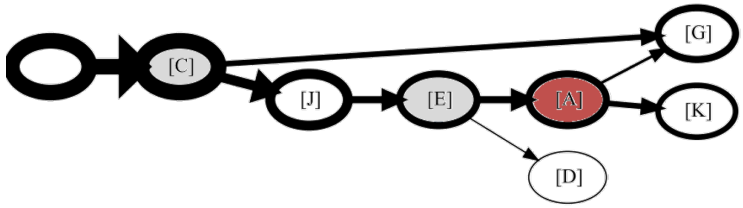
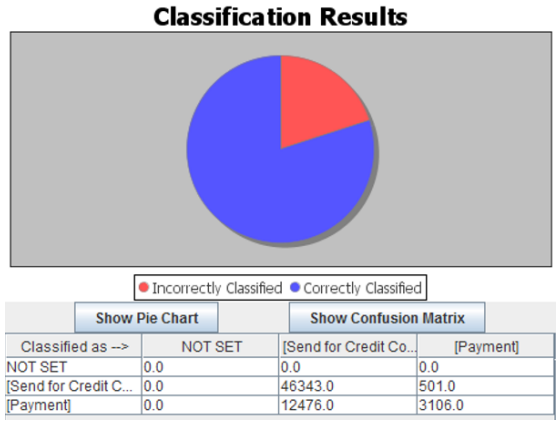


Figure 5.8: Example of an annotated transition system colored with the results of business rules comparison, where the decision points are highlighted in red if their agreement score is below the agreement threshold, and in grey otherwise. States that are not decision points are not highlighted at all.



Classified as -->	NOT SET	[Send for Credit Co...	[Payment]
NOT SET	0.0	0.0	0.0
[Send for Credit C...	0.0	46343.0	501.0
[Payment]	0.0	12476.0	3106.0

Figure 5.9: Representation of a *cell* (i.e., intersection of a row and a column) of a Decision Point Matrix (See Fig. 5.7) that corresponds to the classification results of a set of observation instances (row) using a decision tree (column). These results can be visualized as a pie chart (i.e., correctly and incorrectly classified) or as a confusion matrix.

agreement and disagreement in classifying the instances. The second one is to use a *confusion matrix*, where each column of the confusion matrix represents the instances in a predicted class while each row represents the instances in an actual class. Please note that other ways to present classification results can be used instead.

From this confusion matrix, we can also calculate metrics such as *precision*

and *recall* that can provide insights about the quality of the classifications.

5.3 Implementation

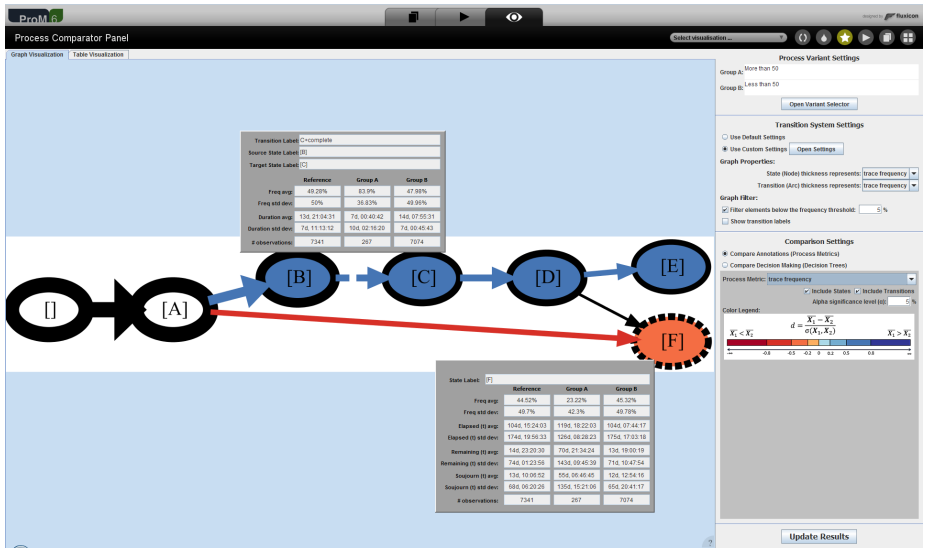
Our approach has been implemented as the *Process Comparator* plugin in the ProM [172] framework, as shown in Figure 5.10.[9]

The tool takes two event logs as input. However, more than two event logs can be compared. This is handled by requesting the user to group these event logs into two groups. Each of these groups is then merged into a single event log and then compared against each other (e.g., "one against the rest" or "one half of the event logs against the other half"). The tool also provides a "hint" functionality to create the two groups of event logs for the users that do not have context knowledge or do not know *a-priori* which event logs to compare. This functionality suggests to compare a single event log against all the others by calculating *similarity scores* between each individual process and the merge of the $n-1$ remaining processes. The similarity score is calculated based on the percentage of elements that present statistically significant differences. Finally, the tool suggests to the user the process that has most differences with the rest as a starting point for comparative analysis.

Our tool allows the user to change state and event representation functions, state and transition measurement functions and several useful parameters (e.g., the significance level of the statistical significance tests) in order to provide flexible representations for the event logs, as shown in Figure 5.10. Our tool also provides frequency filtering capabilities where all the nodes and arcs with lower frequency than a defined threshold will be hidden from the visualization. This allows to filter out rare behavior and to produce clearer visualizations. Also, the elements of the transition system presented as a result are *interactive*. The user can click on any state or transition, and a dialog will pop-up showing either: (1) the values of the annotations of such a state or transition for both event logs (e.g., frequency of occurrence, elapsed time, remaining time, number of traces), as shown in Figure 5.10a, or (2) its decision point matrix if the selected element is a decision point, as shown in Figure 5.10b. The actual content of the dialog will depend on the settings defined by the user (whether they want to compare behavior or business rules).

It is important to note that this implementation relies on the event log being "enriched" (e.g., by the user) with context information in the form of event

[9] The plugin is included in the *ProcessComparator* package in ProM.



(a) Comparison of Behavior

(b) Comparison of Business Rules

Figure 5.10: Screenshot of the *Process Comparator* plugin in the ProM framework. Details are presented in pop-up dialogs when the user clicks on states or transitions showing comparisons according to the defined settings (Compare Behavior or Business Rules).

attributes. Such event attributes enable the extraction of business rules and their comparison.

5.4 Applications

This section describes the application of our tool to two different datasets: a synthetic and a real one. In these applications, the use of the tool is showcased, and the results are interpreted and discussed in order to obtain useful insights.

5.4.1 Using Synthetic Data

This experiment is inspired by the textbook process example: The *loan application* process [50].

This process starts with an *applicant* submitting a loan application. The company checks the application and returns it to the applicant if there are issues with the application. If there are no issues, the company checks the credit history of the applicant, appraises the property (if the loan is for purchasing a property), assesses the risk of the loan, and assesses the eligibility of the application. After this, the company decides whether to approve or reject the loan. If the loan is rejected, the process ends. If the loan is accepted, the company can optionally offer a home insurance quote to the customer, and sends the acceptance pack. Finally, if the company verifies the payment agreement, the application is approved. If payment cannot be verified, the application is cancelled.

Figure 5.11 shows a transition system that represents the control-flow of the loan application process. This transition system was built using the state and activity representation functions $r^s(\sigma) = \#_{activity}(\sigma(|\sigma|))$ and $r^a(e) = \#_{activity}(e)$ (i.e., equivalent to the *directly-follows* graph [156]).

In this experiment, we designed two CPN models of this process that have an identical control-flow structure, but one of the business rules is different.[10] The difference between these two models is that they use different thresholds in the activity "Assess Eligibility" to determine whether a loan is accepted or rejected. In the *high* model, the rejection threshold is set to 7000 euros (i.e., loans of 7000 euros or more are rejected), whereas in the *low* model, the rejection threshold is set to 4000 euros.

[10]The CPN models of both variants can be downloaded from: `http://www.win.tue.nl/~abolt/userfiles/downloads/Models/`

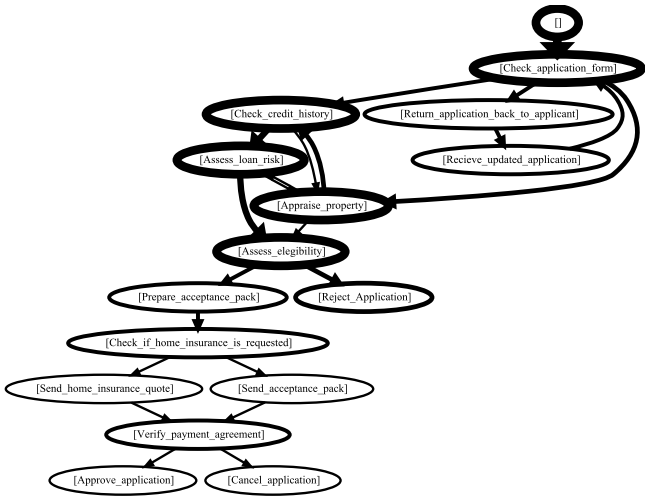


Figure 5.11: Annotated Transition system representing the control-flow of the loan application process. Thickness represents frequency of occurrence.

These two models were used to simulate two event logs (i.e., variants) of 1.000 traces each using CPN Tools: a *high* variant simulated from the *high* model, and a *low* variant simulated from the *low* model[11]. In both event logs, the loan amount of each trace is generated from a uniform distribution between 1.000 and 10.000 euros. We expect to observe a difference between these variants in terms of business rules (due to different rejection thresholds) but also in behavior (i.e., more loans should be rejected in the *low* variant). This is because, in the simulation, the probability of a single randomly-selected loan to be rejected is 0.3 in the *high* variant and 0.6 in the *low* variant.

Figure 5.12 shows the behavior comparison results obtained using our approach. We can note that, indeed, we could detect a statistically significant difference in the percentage of loans that are rejected (highlighted in red): 65% of the loans are rejected (i.e., they reach the *Reject Application* state) in the *low* variant versus 35% in the *high* variant. The blue colors assigned to states and transitions indicate that the part of the process that comes after the acceptance of the loan (i.e., state *Prepare acceptance pack* and subsequent states and transitions) was more frequent in the case of *high* fines. This difference in the

[11] CPN Tools is freely available from: http://cpntools.org

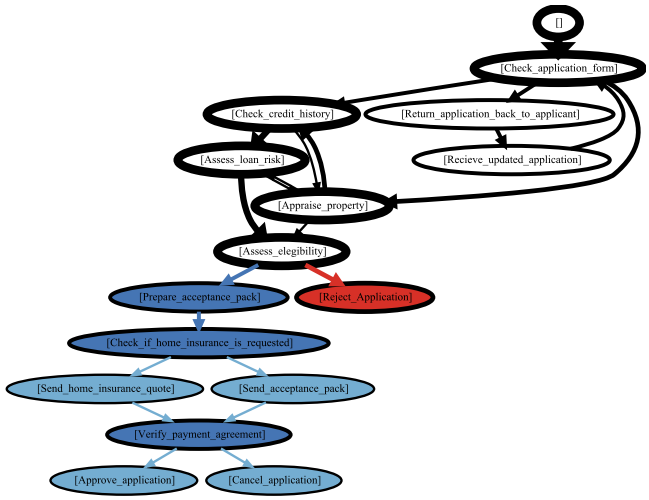


Figure 5.12: Artificial experiment results: Differences in terms of frequency (highlighted in blue and red) were found between the *high* and *low* variants.

percentage of accepted or rejected fines is expected because of the less-strict threshold used in the case of *high* fines.

Figure 5.13 shows the business rules comparison results obtained using our approach. Note that a difference between the two variants in terms of business rules was detected for the decision point *Assess Eligibility*. Figure 5.13 also shows a concrete instantiation of the decision point matrix (see Figure 5.7) for the decision point mentioned above. The elements on this matrix indicate the percentage of a set of instances (defined for each row) correctly classified by a decision tree (defined by each column). Note that even though the self accuracy is perfect (as shown by elements 1 and 4 in Figure 5.13), when the same business rules are applied to the other variant (as shown by elements 2 and 3 in Figure 5.13), the accuracy is not so good.

The decision trees learned for this decision point are shown in Figure 5.14, where Figure 5.14a represents the decision tree learned with the *low* variant and Figure 5.14b represents the decision tree learned with the *high* variant.

Note that in both cases the business rules are very close to the original thresholds (3.993 vs 4.000 for the *low* variant and 6.999 vs 7.000 for the *high* variant).

Figure 5.14c shows the agreement/disagreement decision tree. Note that the

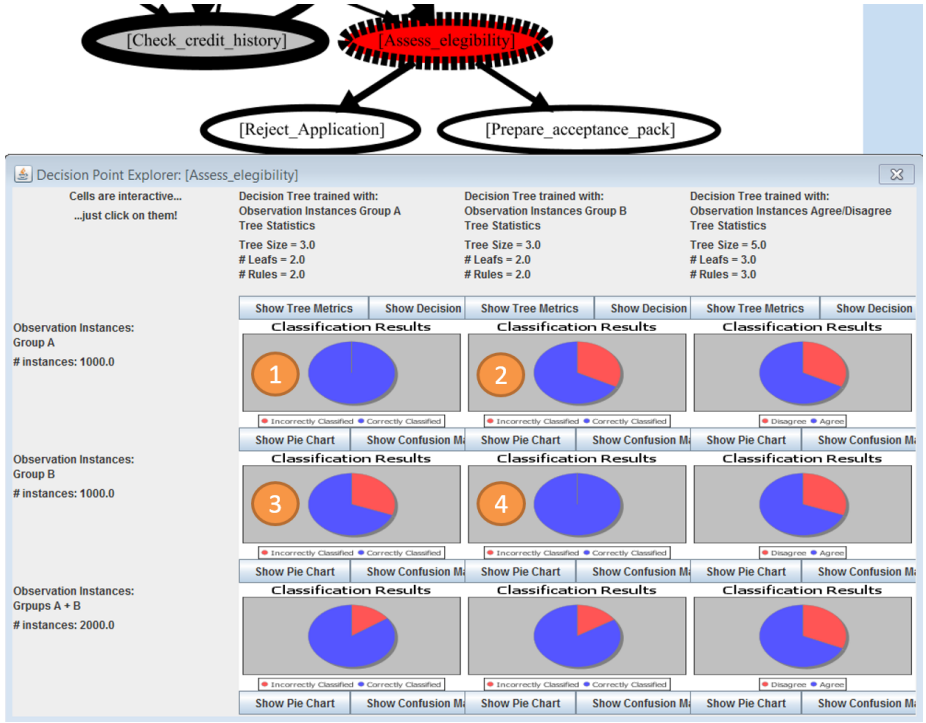
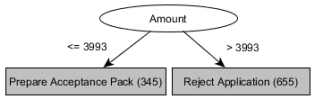


Figure 5.13: Artificial experiment results: Differences in terms of business rules (highlighted in red) were found between the *high* and *low* variants.

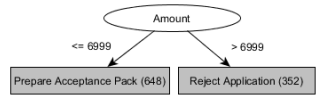
disagreement range is as expected: around the interval from 4.000 to 7.000.

5.4.2 Using Real Data

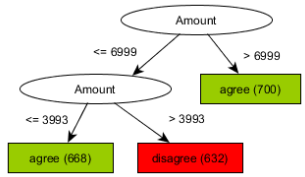
In order to show the usefulness of our approach in practice, we applied our tool to the running example related to the road fines management process (see Section 2.4). For showing the comparison capabilities of our approach, we arbitrarily split the event logs into two sub-logs (i.e., *variants*): the first one contains all the cases where the fine amount was lower than 50 euros (i.e., *low fines*) and the second contains all the cases where the amount of the fine was equal or higher than 50 euros (i.e., *high fines*). Both event logs are annotated with over 20 data attributes. The two event logs were then compared against



(a) Decision tree learned from the *low* variant.



(b) Decision tree learned from the *high* variant.



(c) Agreement/disagreement decision tree.

Figure 5.14: Artificial experiment results: Decision trees learned for the decision point "Assess elegibility". The approach successfully identifies that there is disagreement in the middle range (4.000 - 7.000).

each other in terms of behavior and decision making using our tool in three sets of experiments:

1. The first was based on an abstraction where the last event of the trace is considered. Given an event log L, a trace $\sigma \in P_L$ and an event $e \in E_L$, we used the state and transition abstractions: $r^s(\sigma) = \#_{activity}(\sigma(|\sigma|))$ and $r^a(e) = \#_{activity}(e)$. As a measurement for comparison, the elapsed time was used as defined in Eqs. 5.2 and 5.4, thus comparing the time differences when activities were executed.

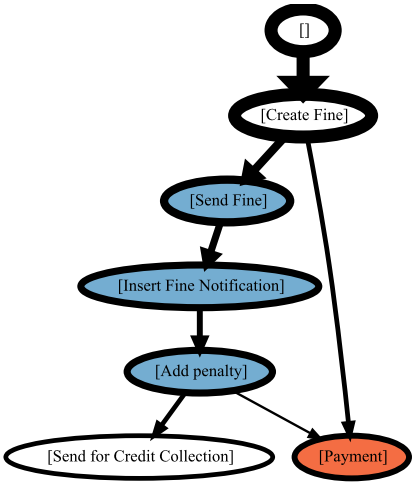
2. The second experiment was based on the same abstractions as the first experiment (i.e., identical state and transition representation functions). However, the focus of this experiment is to make comparisons in terms of decision-making.

3. In the third experiment, the last two events of the trace were considered, as we used $r^s(\sigma) = \langle \#_{activity}(\sigma(|\sigma|)), \#_{activity}(\sigma(|\sigma| - 1)) \rangle$ and $r^a(e) = \#_{activity}(e)$. Unlike the previous two experiments, we used this abstraction in this experiment to include more than just directly-follows relations. The occurrence measurements for comparison were used as defined in Eqs. 5.1 and 5.3.

In the first and last experiments (i.e., comparison in terms of behavior), we used a confidence level $\alpha = 0.05$ for the statistical significance tests. The results of the three experiments mentioned above are described in the remainder of this section.

First Experiment: Differences in Performance

Figure 5.15 shows the results of the first experiment, where many relevant performance differences were detected. Figure 5.15a shows a transition system annotated with the significant differences found in terms of *elapsed time*. Figure 5.15b shows information popups obtained after clicking on the states *Send Fine* and *Payment*. Such popups indicate the average and spread (i.e., standard deviation) of several control-flow and performance annotations. Red colors are



(a) Annotated transition system showing performance differences. Blue shades represent earlier executions of activities or reaching of states in *high* fines. Red shades represent the other way around. White indicates that no significant differences can be observed.

State Label: [Send Fine]

	Reference	Group A	Group B
Freq avg:	69.15%	66.54%	85.07%
Elapsed (t) avg:	87d, 12:20:54.54	90d, 16:02:37.50	72d, 12:29:24.74
Elapsed (t) std dev:	42d, 11:47:43.72	41d, 10:34:31.33	44d, 04:34:35.40
Remaining (t) avg:	401d, 22:57:09.7	396d, 00:36:54.1	430d, 03:37:53.3
Remaining (t) std dev:	319d, 14:00:00.6	316d, 17:30:28.0	331d, 09:02:07.7
Soujourn (t) avg:	14d, 23:17:03.70	14d, 19:48:44.37	5d, 15:47:26.015
Soujourn (t) std dev:	23d, 02:53:15.15(	23d, 07:17:44.69	22d, 05:08:18.78
# observations:	150370	129127	21243

State Label: [Payment]

	Reference	Group A	Group B
Freq avg:	46.36%	48.51%	33.31%
Elapsed (t) avg:	105d, 04:46:43.2	99d, 22:09:43.18	151d, 07:55:31.9
Elapsed (t) std dev:	172d, 20:36:57.4	169d, 10:21:08.4	194d, 08:26:43.3
Remaining (t) avg:	25d, 10:06:26.74	23d, 08:42:06.50	43d, 10:06:51.59
Remaining (t) std dev:	97d, 03:46:06.88	92d, 22:32:49.26	126d, 20:24:17.5
Soujourn (t) avg:	16d, 20:52:34.21	15d, 06:42:54.07	30d, 18:33:37.76
Soujourn (t) std dev:	77d, 10:57:05.79	73d, 02:44:30.96	107d, 05:58:08.1
# observations:	150370	129127	21243

(b) Information popups obtained from clicking the states `Send Fine` and `Payment` in the Process Comparison Tool. Group A refers to *low* fines and Group B refers to *high* fines.

Figure 5.15: Performance (*elapsed time*) comparison between *high* and *low* fines.

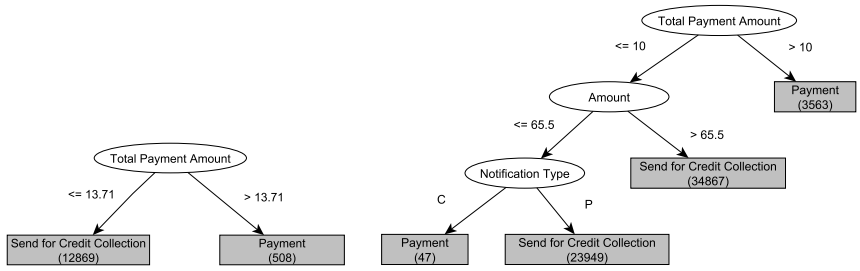
assigned to states and transitions that are reached or executed statistically significantly earlier in *low* fines, whereas blue colors are assigned when the opposite occurs.

The red color assigned to state *Payment* indicates that payments were received significantly earlier for low fines (99 days versus 151 days). Conversely, the blue color assigned to the state *Send Fine* indicates that *high* fines are sent to offenders significantly earlier (72 days versus 90 days). The fact that the *Create Fine* state is white indicates that there is no statistically significant difference in how early *Create Fine* is executed.

Second Experiment: Differences in Business Rules

The results of the second experiment showed the states *Create Fine* and *Add Penalty* were identified as decision points. After a fine has been created, it can be immediately paid (i.e., *Payment*). If it is not fully paid, it is sent to the offender (i.e., *Send Fine*). Since the decision in this case is trivial, we proceeded to analyze the *Add Penalty* decision point in more detail.

From this decision point, a fine can be either paid or sent to credit collection (i.e., states *Payment* and *Send to Credit Collection* respectively). Figure 5.16 shows the decision trees of both process variants for this decision point. We can



(a) $DT^1_{\text{Add penalty}}$ (b) $DT^2_{\text{Add penalty}}$

Figure 5.16: Decision Trees of variants *high fines* (DT^1) and *low fines* (DT^2) for the decision point *Add Penalty*. The leaf nodes (i.e., nodes without child) also show the number of instances classified into them (in brackets).

observe that both decision trees follow different decision making rules. From all the data attributes contained in the event log, the one that is most determinant

to predict which activity will follow is *totalPaymentAmount*. This data attribute contains the accumulation of all the payments for that fine.

Figure 5.16 shows that if the *totalPaymentAmount* is higher than 13.7 (for the high fines) or 10 (for the low fines), both decision trees predict that the fine will be paid. This means that if an offender has partially paid a given amount and a penalty is added (increasing the fine *Amount* to be paid) most offenders will pay the fine. In the case of the high fines, if the *totalPaymentAmount* is lower or equal to 13.7, the fine is most likely to be sent for credit collection. This includes all the offenders that have not paid any part of their fine, and it seems that the penalty added does not stimulate the offender to pay the fine. In the case of low fines, if the *totalPaymentAmount* is lower or equal to 10 and if the new amount of the fine after the penalty is higher than 65.6, it will be most likely sent to credit collection. On the other hand, if the new amount is lower or equal to 65.6 and if the *notificationType* is "C" (i.e., *Conducente*, which is the Italian for "Driver"), means that the notification was sent to the driver of the car when the fine was given, and it leads to the payment of the fine. However, if the *notificationType* is "P" (i.e., *Proprietario*, which is the Italian for "Owner"), means that the notification was sent to the owner of the car, and it most likely leads to the fine being sent for credit collection.

We also analyzed how each decision tree can be used to replace the other, adapting their decision making criteria. Figure 5.17 shows that, although the decision trees are different, they tend to make the same decisions.

This can be observed in the first row of pie charts of Figure 5.17; The pie charts indicate a similar classification accuracy. From the above, we can conclude that the decision making in both process variants is similar, even though their decision making is structurally different (i.e., different decision rules).

So far, we have concluded that the decision making in the studied variants is very similar in the *Add Penalty* decision point. However, we are also interested in analyzing under which conditions these trees disagree in their predictions. For this purpose, we built a decision tree with the extended observation instances of both variants to analyze the decision rules that lead to agreements or disagreements (as discussed in Section 5.2.2). The resulting tree is shown in Figure 5.18. This similarity tree confirms the conclusions obtained from the cross classification mentioned above: in 99.93% out of 75803 cases, the predictions of the previous decision trees are the same (i.e., *agree*). However, the cases where they disagree are characterized, in 47 out of 53 cases, by the notification being sent to the driver (*notificationType* = "C") and the amount after the fine (*Amount*) is less or equal to 65.6 euros. This indicates that, even though the business rules of both variants are structurally different in the [Add Penalty]

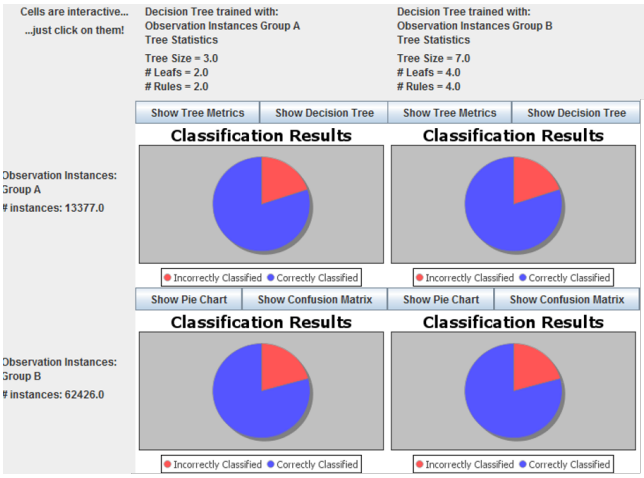


Figure 5.17: Section of the Decision Point Matrix for the decision point *Add Penalty*. Each pie chart shows how each decision tree (column) classifies sets of observation instances (rows). Group A corresponds to high fines and Group B to low fines.

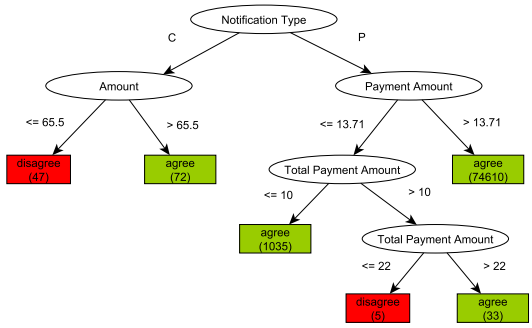


Figure 5.18: Decision tree that classifies extended observation instances into whether the trees of each variant agree (green) or disagree (orange) in their classifications in the decision point [Add Penalty]. The leaf nodes (i.e., nodes without child) also show the number of instances classified into them (in brackets).

decision point, they agree in most cases.

Third Experiment: Differences in Control-flow Frequency

Figure 5.19 illustrates the output of the third experiment. Orange shade ovals and arcs represent states reached or transitions executed significantly more often in low fines compared with high fines. Blue shades refer to the opposite. The first observation is that low fines are usually immediately paid without requiring the local police to send a copy of fine to the offender. This can be seen through the orange-colored state *[Payment,Create Fine]* and the transition from

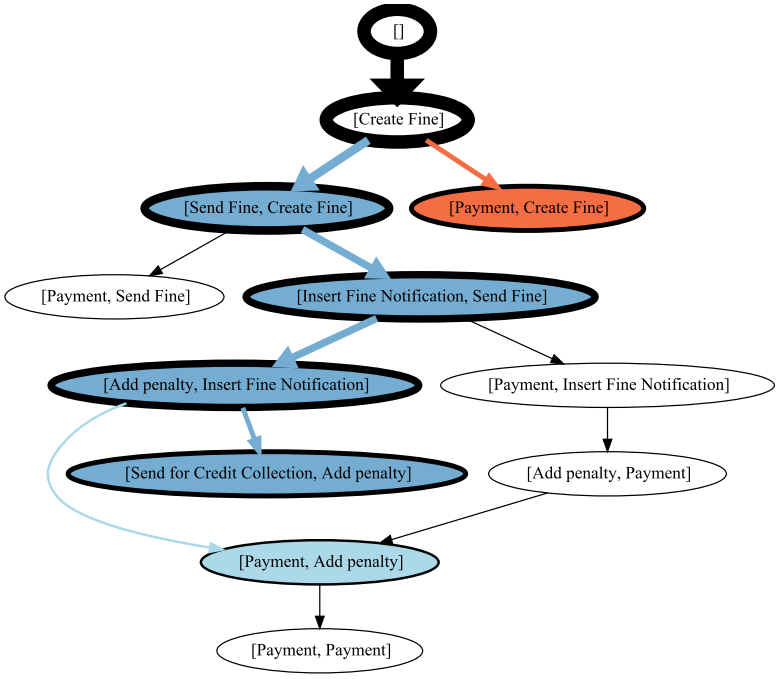


Figure 5.19: *Occurrence* frequency comparison. Colored states (i.e., nodes) and transitions (i.e., edges) contain statistically significant differences between the two event logs. Blue nodes and arcs show a higher fraction of cases involving a high fine. Orange nodes and arcs signal a higher fraction of cases involving a low fine.

[Create Fine] to this state. Conversely, high fines are more often sent to the offender than low fines, as one can observe through the blue-colored state *[Send Fine, Create Fine]*. Similar observations can be derived by looking at the other states and transitions.

Figure 5.19 also indicates that, for *low* and *high* fines, it is similarly frequent that offenders perform incomplete payments, which causes a penalty to be added[12], which are subsequently followed by a second payment to probably complete the fine payment. This can be observed in the white-colored states *[Payment,Insert Fine Notification]* and *[Add Penalty, Payment]*. Note that, for high fines, it is significantly more frequent that a payment only occurs after adding the penalty. This can be seen from the blue color associated with the transition between states *[Add Penalty, Insert Fine Notification]* and *[Payment,Add Penalty]*. Please observe that the latter finding could not be observed if we used an abstraction solely based on the last occurred event.

5.5 Conclusions

The problem of comparing process variants is highly relevant. Many companies are observing that the executions of their processes are not always optimal and subject to variations. There may be interesting differences between departments, customer groups, and periods. Processes may change because of the influence of several factors, such as the period of the year, the geographical location of the process execution or the resource unit in charge. Some recent approaches aim to compare the execution of different process variants. However, existing approaches tend to focus on the control-flow perspective and often detect differences that are statistically insignificant.

To our knowledge, none of the existing approaches is able to detect the relevant (i.e., statistically significant) behavioral differences between process variants in terms of any annotation (e.g., performance) or business rules based on their recorded event logs. To address this challenge, we developed a new technique based on transition systems that detects statistically significant differences between process variants in terms of any measurement annotations, and shows the similarities and differences of the business rules between them, using event logs as input. We used transition systems to avoid being dependent on a particular discovery algorithm and representation. Transition systems can

[12]According to the Italian laws, if a fine is not paid in full within 90 days, a penalty is added so that the due amount doubles.

be created using different perspectives (e.g., next to control-flow one may also consider resource interaction) and a range of parameters can be used to control the result.

Our implementation is provided with concrete annotations, which are related to the control-flow frequency (named *occurrence*) and to the time perspective (*elapsed time, remaining time*, and *sojourn time* annotations). However, the framework is extensible and allows users to easily add new measurement functions. This extensibility enables the user to use context information (e.g., workload, weather) in the extraction and comparison of business rules.

The applications showed the approach enables users to pinpoint differences that previous approaches failed to provide. Also, our approach does not show differences that are statistically insignificant, which are conversely returned by other approaches.

As a limitation, we would like to discuss the choice of transition systems as the representation used for process models in this chapter. Transition systems have some limitations: they can easily explode into giant state-spaces, they cannot represent concurrency properly, etc. However, transition systems are what is considered in the machine learning field as a "stable classifier" (i.e., small changes in the data have small effects in the model). In contrast, Petri nets, BPMN and other higher level representations that can handle concurrency have a tendency to be "unstable classifiers" (i.e., small changes in the data can have large effects in the model). For example, removing a single trace from an event log may have a significant effect in a Petri net structure (e.g., border cases around the discovery technique´s parameter thresholds), yet it will have a low impact on the structure of a transition system.

Chapter 6

Process Variant Detection

Business processes are not static: They have to adapt to constant environment changes (e.g., customer preferences, legal regulations, new competitors). Like any live species, companies (and their business processes) also evolve according to Darwinian evolution: "The best to adapt is the one that thrives". It is not uncommon for companies that the same business process has to adapt to different contexts simultaneously, which leads to variability in the behavior of such processes.

As discussed in Section 1.2, process variability is not only related to the *control-flow* perspective (e.g., a process may skip risk assessment steps for gold customers), but can also be related to other perspectives, such as *performance*. For example, if two branches of a company execute their processes in the same way (i.e., same control-flow) but there are huge performance differences between the branches, it is interesting to understand and explain such differences. Most process *discovery* techniques (i.e., discovering process models from event logs) deal with process *control-flow* variability by combining all the observed *executions* (i.e., cases) of a process into a single process model. This results in what is known as *spaghetti models* (i.e., illegible process models). Note that process variability can be related to any context information, as long as it is presented in the form of data attributes of events.

Variability makes processes more difficult to analyze and understand. In this chapter, we deal with variability in processes by identifying *process variants* (see Definition 2.13) in event data, which are then enriched with event attributes that encode variant information. This enriched event data can also be split

into process variants using a *variant* dimension in a process cube, created from the newly-created attributes (see Chapter 3) and later, such variants can be used to run process mining workflows (see Chapter 4) or to be compared with each other using process comparison techniques such as the one presented in Chapter 5, as illustrated in Figure 6.1.

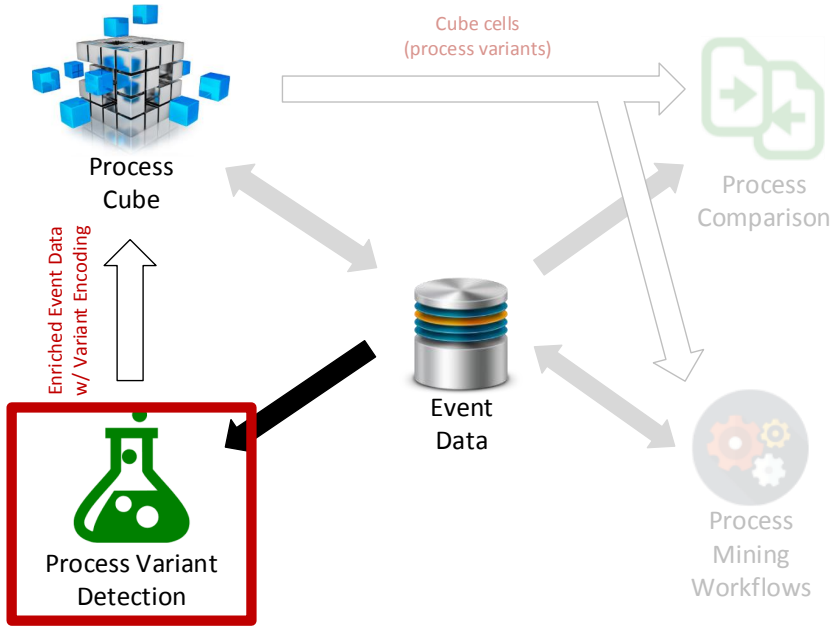


Figure 6.1: Overview of the scope of this chapter: event data is analyzed and process variants are found. This information is used to enrich the log with new event attributes that explicitly mention the variant it belongs to. These new variant-related attributes can be used in a process cube for splitting event data into such process variants. Unused interactions are greyed out.

In this chapter, we propose a technique to detect relevant process variants in an event log using the control-flow, performance, and context attributes of events in an interactive and exploratory way, where only *relevant* results are presented. To achieve this, we leveraged on well-proven data mining techniques

for Recursive Partitioning driven by Conditional Inference (RPCI) over event attributes.

It is important to note that the type of analysis performed with our approach can also be achieved by combining other approaches and standard data mining techniques. However, such techniques require extensive and manual ad-hoc parametrization and configuration to achieve the same results that our approach can obtain in a much easier way. We achieve this by discovering a process model from an event log and using such model to identify points of interest in the process (e.g., a given *state* in the process). Then, the same variability analysis is automatically performed in each point of interest and the summarized results for the whole process are presented to the user as result.

The remainder of this chapter is structured as follows. Section 6.1 discusses related work. Section 6.2 introduces our approach and discusses how process variants can be identified in event logs with the help of process models. Section 6.3 describes the implementation of our approach. Section 6.4 describes the application of our tool using real event data. Finally, Section 6.5 concludes the chapter.

6.1 Related Work

The identification of process variants can be related to the well-known *trace clustering* problem. In recent years, several approaches for trace clustering have been proposed in literature. Even though these approaches may use different techniques, they all have a common goal: Split an event log into smaller event logs with less variability.

Existing trace clustering approaches can be grouped into four categories, based on how clusters are obtained:

- Structural similarity
- Concept drift detection
- Performance analysis
- Attribute correlation

Trace clustering techniques based on *structural similarity* [21, 139, 173, 188] focus on clustering traces based on their control-flow structure (i.e., sequence of activities). A downside of techniques based on structural similarity is that, even though the discovered clusters contain traces with similar control-flow,

such clusters are often difficult to characterize using data attributes (e.g., type of customer, amount of a loan request). For example, all the traces in an event log that executed the same sequence of activities can be trivially clustered together, but such cluster is meaningless from a business perspective if it cannot be characterized using data attributes (e.g., most traces in the cluster can be related to *vip* customers).

Trace clustering techniques based on *concept drift detection* focus on a specific type of process variability i.e., control-flow variability over time. An extensive survey on concept drift techniques in data mining is presented in [56]. Concept drift detection techniques have been applied in process mining by several authors [112, 116, 135]. These approaches focus on detecting points in time when the control-flow of a process changed. When a change point is detected, an event log can be split based on whether traces were executed before or after such point in time. The downside of these techniques is their limited scope (i.e., control-flow changes over time), ignoring other perspectives such as performance or other data attributes.

Trace clustering techniques based on *performance analysis* [67, 68] focus on detecting differences in the performance of the process and characterizing them by using control-flow, performance, and context attributes. Then, traces with similar performance can be grouped together, and clusters can be characterized using other perspectives (e.g., control-flow, other data attributes). The downside of these techniques is also their limited scope, as they only cluster traces based on their performance.

Trace clustering techniques based on *attribute correlation* [45, 74] are more general, as they aim to group cases depending on any event or trace attributes. The approach proposed in this chapter falls in this category and is closely related to [45], which focuses on classifying specific selections of events by building decision or regression trees with the attributes of such events that are later used to classify traces into process variants.

6.2 Process Variant Detection

The approach presented in this chapter allows for grouping traces of an event log into *process variants* (see Definition 2.13). The overview of our approach and its steps are illustrated in Figure 6.2. The approach consists of the following steps:

Step 1: Given an event log, a transition system is created.

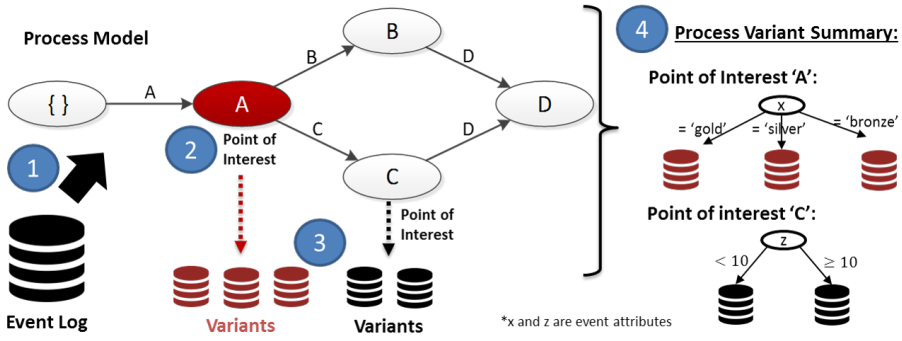


Figure 6.2: Overview and steps of our approach to detect process variants in event logs.

We use transition systems as process models because any process model (in any notation) that has executable semantics can be translated into a transition system. Section 2.3.4 discussed how a transition system can be built from an event log based on state and activity abstractions. Given an event log L, a state representation function r^s and an activity representation function r^a, the output of this step is a *transition system* $TS^{(r^s,r^a,L)} = (S, A, T)$.

Step 2: Points of interest are identified in the transition system.
Informally, a point of interest can be any state or transition of the transition system. Users can select points of interest in many ways, e.g., the most frequently visited states or the transitions corresponding to a specific part of the process. The output of this step is a *set of points of interest*. Each point of interest is related to a set of traces that reach it. This is discussed in detail in Section 6.2.1.

Step 3: For each point of interest, the set of traces that reach it is partitioned into process variants.
Given a point of interest and a set of traces that reach it, the objective of this step is to partition such set of traces into non-overlapping sets of traces, i.e., process variants. This is performed for *all the points of interest* defined in the previous step. The details of this step are discussed in Section 6.2.2.

Step 4: A summary of process variants is produced, where the splitting criteria and the resulting variants are shown for each point of interest.

To simplify the interpretation of the results of the previous step, we present a summary of *only the points of interest where process variants were found* (see Section 6.3). For each point of interest, the splitting criteria are clearly presented, and the obtained process variants are used to enrich the original event log with variant information by explicitly encoding it as data attributes, so it can be later split using process cubes or used by other process mining techniques.

Step 1 (creating a transition system from an event log) was discussed in detail in Sections 2.3.4 and 5.2.1. Hence, we assume that the reader is already familiar with it.

Therefore, the remainder of this section will discuss steps 2 and 3 of the approach in detail.

6.2.1 Defining Points of Interest in a Transition System

The second step in our approach (step 2 in Figure 6.2) is to identify points of interest in the transition system obtained from step 1. Given an event log L and a transition system $TS^{(r^s,r^a,L)} = (S, A, T)$, any state $s \in S$ and any transition $t \in T$ can be a *point of interest*.

A point of interest can be related to a set of traces that "reach" it. Given an event log L and a transition system $TS^{(r^s,r^a,L)} = (S, A, T)$, every potential point of interest $p \in S \cup T$ can be related to a set of traces in the event log through the function $Tr^{(r^s,r^a,L)} : (S \cup T) \to \mathbb{P}(L)$. This function is defined as:

$$Tr^{(r^s,r^a,L)}(p) = \begin{cases} \bigcup_{\sigma \in L} \{\sigma | \exists_{\sigma' \in pref^{\diamond}(\sigma)} : r^s(\sigma') = s\} & \text{if } p \in S \\ \bigcup_{\sigma \in L} \{\sigma | \exists_{\sigma' \in pref^{\diamond}(\sigma) \setminus \{\langle\rangle\}} : r^s(pref^{|\sigma'|-1}(\sigma')) \\ \qquad = s_1 \wedge\ r^a(\sigma') = a \wedge r^s(\sigma') = s_2\} & \text{if } p = (s_1, a, s_2) \in T \end{cases}$$

All the states and transitions of a transition system can be initially considered as points of interest. However, we select a subset of points of interest $PI \subseteq S \cup T$ because not all points of interest are equally *relevant*. The relevance of a point of interest with respect to an event log is defined as follows.

Definition 6.1 (Relevance of points of interest)**.** *Let L be an event log and $TS^{(r^s,r^a,L)} = (S, A, T)$ be a transition system. The* relevance *of a point of interest $p \in S \cup T$ is defined as $|Tr^{(r^s,r^a,L)}(p)|/|L|$ i.e., the percentage of traces in the event log that reach that point of interest.*

For example, all the states and transitions that are reached by less than 5% of the traces in the event log could be considered as irrelevant because they rarely happen, and can be removed from the analysis. In the implementation of our approach (see Section 6.3), a *threshold of relevance* is defined by the user, which defines the minimum allowed relevance of points of interest, which is used to filter out points of interest with a lower relevance than the threshold. However, other alternative notions of relevance can be defined, and the users can arbitrarily select points of interest from $S \cup T$. For example, if there is a *bottleneck* in the process, users can select the points of interest related to the states and transitions where such bottleneck is located.

The output of this step is a set of points of interest $PI \subseteq S \cup T$, where each point of interest $p \in PI$ is related to the set of traces $Tr^{(r^s,r^a,L)}(p)$.

6.2.2 Finding Variants in a Point of Interest

The third step in our approach (step 3 in Figure 6.2) is to find process variants in the points of interest defined above. As described in Definition 2.13 in Section 2.2, process variants are sets of traces. The objective of this step is, for each point of interest p (e.g., a state or transition), to group the set of traces that reach it (i.e., $Tr^{(r^s,r^a,L)}(p)$) into *process variants* by leveraging on their event attributes.

It is important to note that traces can reach a point of interest more than once (e.g., in the presence of loops), hence, can have multiple values for the same event attribute. However, because we need single values for event attributes, we must choose only one event within the trace in order to use its attribute values for such cases. In this chapter, we choose the last event of the shortest prefix of a trace that reaches the point of interest i.e., the *first* event if observed multiple times. For this purpose, we introduce a helper function σ_p that, given a trace σ and an event $e \in \sigma$, returns the prefix of σ that ends with e: $\sigma_p(e) = \sigma' \in pref^{\diamond}(\sigma)$ such that $\sigma'(|\sigma'|) = e$. We use this helper function to define, for any point of interest, the set of events related to traces that reach it. This set of events will be used later for creating *instances* that can be clustered or partitioned.

Definition 6.2 (Set of events related to traces that reach a point of interest)**.** *Let L be an event log, $E_L = \{e \in \sigma | \sigma \in L\}$ be the set of events in traces of L, $TS^{(r^s,r^a,L)} = (S, A, T)$ be the corresponding transition system, $p \in PI \subseteq S \cup T$ be a point of interest and $Tr^{(r^s,r^a,L)}(p)$ be the set of traces of L that reach p. We define the function $Ev^{(r^s,r^a,p,L)} : Tr^{(r^s,r^a,L)}(p) \rightarrow E_L$ that maps traces to the first*

event within that trace with which p is reached. For any trace $\sigma \in Tr^{(r^s,r^a,L)}(p)$, the function Ev is defined as:

$$Ev^{(r^s,r^a,p,L)}(\sigma) = \begin{cases} e \in \sigma \Big| r^s(\sigma_p(e)) = p \wedge \\ \forall_{e' \in pref^{|\sigma_p(e)|-1}(\sigma)} r^s(\sigma_p(e')) \neq p \end{cases} \quad \textit{if } p \in S$$

$$Ev^{(r^s,r^a,p,L)}(\sigma) = \begin{cases} e \in \sigma \Big| r^s(pref^{|\sigma_p(e)|-1}(\sigma)) = s_1 \wedge \\ r^s(\sigma_p(e)) = s_2 \wedge r^a(e) = a \wedge \\ \Big(\forall_{e' \in pref^{|\sigma_p(e)|-1}(\sigma))} r^s(\sigma_p(e')) \neq s_2 \\ \vee\ r^a(e') \neq a \\ \vee\ r^s(pref^{|\sigma_p(e')|-1}(\sigma)) \neq s_1\Big) \end{cases} \quad \textit{if } p = (s_1, a, s_2) \in T$$

In order to partition a set of traces into subsets of traces, we need to first translate them into *instances* that can be used by classification and partitioning techniques.

Given a set of traces $Tr(p)$ related to a point of interest p, we define the set of available attributes as: $A^{(r^s,r^a,L,p)} = \bigcup_{\sigma \in Tr^{(r^s,r^a,L)}(p)} atts\big(Ev^{(r^s,r^a,p,L)}(\sigma)\big)$.[1]

Analogously to several machine learning techniques, instances need a *dependent variable* $d \in A^{(r^s,r^a,L,p)}$ to be defined (a.k.a, *class* or *label*). The dependent variable d represents the outcome whose variation is being studied, and it will determine the process variants that are found. For example, if one wants to detect *control-flow variants* in a process (i.e., variants with differences in control-flow), then the dependent variable should be a control-flow event attribute of the process (e.g., the *next activity* to occur). Alternatively, if one wants to detect *performance variants* in a process (i.e., variants with differences in the performance e.g., *slower* and *faster* variants), then the dependent variable should be an event attribute of the process that encodes the performance level e.g., the *elapsed time* within a trace. See Definition 2.11 in Section 2.2 for examples of these attributes obtained with trace manipulation functions.

The remaining $A^{(r^s,r^a,L,p)} \setminus \{d\}$ attributes are considered as *independent variables* which can be used to split the instances in order to reduce the variability of the dependent variable in the resulting partitions i.e., process variants. After defining the dependent and independent variables, we can build instances from traces.

[1] Recall that the function $atts$ maps events to a set of attributes related to them (see Section 2.2).

Definition 6.3 (Set of Instances related to a point of interest)**.** *Let L be an event log, $TS^{(r^s,r^a,L)} = (S, A, T)$ be a transition system, $p \in PI \subseteq S \cup T$ be a point of interest, $Tr^{(r^s,r^a,L)}(p)$ be the set of traces of L that reach p, $d \in A^{(r^s,r^a,L,p)}$ be a dependent attribute and $A^{(r^s,r^a,L,p)} \setminus \{d\} = \{a_1, ..., a_n\}$ be the set of independent attributes. The set of instances related to p is defined by the function $In : PI \rightarrow \mathbb{P}((\mathcal{V} \cup \{\perp\})^{|A^{(r^s,r^a,L,p)}|} \times L)$, where for any point of interest $p \in PI$, the set of instances related to it is defined as:*

$$In^{(r^s,r^a,L)}(p) = \bigcup_{\sigma \in Tr^{(r^s,r^a,L)}(p)} \Big\{ \Big((\#_{a_1}(Ev^{(r^s,r^a,p,L)}(\sigma)), \ldots \\ \ldots, \#_{a_n}(Ev^{(r^s,r^a,p,L)}(\sigma)), \#_d(Ev^{(r^s,r^a,p,L)}(\sigma))), \sigma \Big) \Big\}$$

Note that, differently from Definition 5.8, every trace in $Tr(p)$ is mapped to a single instance. An instance in $In(p)$ is composed by a list of $|A^{(r^s,r^a,L,p)}| - 1$ independent variable values (i.e., $\#_{a_1}(Ev(\sigma))$ to $\#_{a_n}(Ev(\sigma))$), a dependent variable value or *class* (i.e., $\#_d(Ev(\sigma))$), and the trace it was generated from (i.e., σ).

Also note that, unlike Definition 5.8, in this chapter $In(p)$ is not a multiset of instances because every instance is unique since it incorporates the trace, which is unique being composed by unique events.

Most classification and partitioning techniques will only use the first two elements of our instances (i.e., independent and dependent attribute values). The third element (i.e., the trace) is used to simply identify the trace from which the instance was generated from. This becomes useful after partitioning the instances, where each resulting subset of instances (i.e., a partition) can be directly related to a set of traces (i.e., a process variant).

The following example shows how a set of instances can be obtained for a point of interest p using the running example of this thesis.

Example 6.1 (Instances related to a Point of Interest)**.** Let's consider the following transition system (Figure 6.3) related to the road fines management process introduced in Section 2.4:

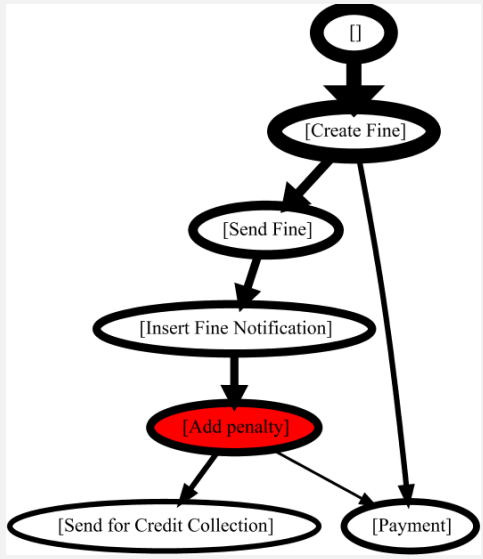


Figure 6.3: Transition system representing the behavior of the road fines management process.

This transition system was built using the state representation function $r^s(\sigma) = \#_{activity}(\sigma(|\sigma|))$ and the activity representation function $r^a(e) = \#_{activity}(e)$. Note that this transition system has 6 states and 7 transitions which can be points of interest. Also note that the empty state $r^s(\langle\rangle)$ is not considered as a point of interest because it does not have events related to it (see the discussion in Section 2.3.4).

For the remainder of this example, we will focus on the state *Add Penalty* (highlighted in red) as our point of interest p. Let's now consider the following four traces:

Table 6.1: Selection of four traces of the road fines management process

event id	**fine id**	activity	timestamp	amount	next activity	elapsed time
001	A1	create fine	24/07/2006	35	send fine	0
002	A1	send fine	05/12/2006	35	-	134 days
003	A100	create fine	02/08/2006	35	send fine	0
004	A100	send fine	12/12/2006	35	insert fine notification	132 days
005	A100	insert fine notification	15/01/2007	35	add penalty	166 days
006	A100	add penalty	16/03/2007	71.5	send for credit collection	226 days
007	A100	send for credit collection	30/03/2009	71.5	-	972 days
008	A10000	create fine	09/03/2007	36	send fine	0
009	A10000	send fine	17/07/2007	36	insert fine notification	131 days
010	A10000	insert fine notification	02/08/2007	36	add penalty	147 days
011	A10000	add penalty	01/10/2007	74	payment	207 days
012	A10000	payment	09/09/2008	74	-	551 days
013	A10005	create fine	20/03/2007	36	payment	0
014	A10005	payment	21/03/2007	36	-	1 day

Note that the *fine id* field indicates the different traces: In this example, a fine corresponds to a trace. From these four traces, only traces A100 and A10000 reach the point of interest $p =$ `Add Penalty`. This means that such traces are elements of $Tr^{(r^s, r^a, L)}(p)$. Note that since traces A1 and A10005 do not reach the state *Add Penalty*, they will not be considered for this point of interest.

Alternatively, if the point of interest to be analyzed was the state `Create Fine` instead, all four traces described above would be considered, as they all reach such point of interest.

As discussed before, for each trace that reaches p, we need to select the event in the trace that first reached the point of interest $p =$ *Add Penalty*. For the trace "$A100$" we select the event "006", and for the trace "$A10000$" we select the event "011". From each of these traces, an *instance* is built. In this example, we consider the *elapsed time* as the dependent variable i.e., the *class* of the instance.

Table 6.2 shows the two instances that can be obtained from the traces described above in Table 6.1.

Table 6.2: Instances obtained from the traces described in Table 6.1 when they reach the point of interest `Add Penalty`.

activity	timestamp	amount	next activity	**elapsed time** (dependent)	**fine id** (trace)
add penalty	16/03/2007	71.5	send for credit collection	226 days	A100
add penalty	01/10/2007	74	payment	207 days	A10000

Note that for each instance, the first four columns correspond to the values of the independent variables related to the trace σ. The fifth and sixth

column correspond to the elapsed time (i.e., the dependent variable) and the id of trace σ respectively. Also note that, formally, an instance contains a whole trace, but for the sake of simplicity we only mention its trace id. These instances can now be used to detect process variants in the point of interest $p =$ *Add Penalty*.

The set of instances obtained for any point of interest (i.e., $In^{(r^s,r^a,L)}(p)$) can now be partitioned into process variants by leveraging on its event attribute values. Many clustering algorithms can be uses to partition such sets of instances e.g., classification and regression trees such as C4.5 [125], or any other non-overlapping clustering technique.

In this chapter, we use a technique named *Recursive Partitioning by Conditional Inference* (RPCI) [69] to partition sets of instances related to each point of interest. The remainder of this section is described as follows. First, we discuss the details of this specific choice, and present an overview of how RPCI works. Finally, we describe how to use sets of instances related to specific process variants to enrich event data with variant-related attributes.

Recursive Partitioning by Conditional Inference

Unlike other classification techniques (e.g., classification and regression trees) RPCI provides an unbiased selection and binary splitting mechanism by means of *statistical tests of independence* between the independent variables and the dependent variable.

One of the main differences is that decision and regression trees aim to predict a value, whereas RPCI aims to partition data. Hence, differences in minority classes would be overseen by approaches based on decision and regression trees. For example, consider an activity X which can be followed by A, B or C. Decision-tree-based approaches (e.g., the work presented in [45]) would not find a difference between (A = 60%, B = 40%, C = 0%) and (A = 60%, B = 0%, C = 40%) because in both cases the majority class, hence, the actual predicted value, is A (60% of the cases). RPCI detects this because it focuses on the differences between the distributions instead of the expected (i.e., most probable) values.

RPCI is based on the work by Strasser and Weber [143], which defines independence tests based on permutations using the asymptotic properties of linear statistics derived from arbitrary distributions. The specific independence tests used depend on the distributional characteristics of the dependent and independent variables. These independence tests are concretely used to determine

attribute correlations and optimal splittings of the set of instances.

In a nutshell, RPCI is described for a set of instances $In^{(r^s, r^a, L)}(p)$ (denoted as I) in Algorithm 1. The algorithm takes a set of instances as input and returns partitions as output in the form of a set of sets of instances.

Algorithm 1 Recursive Partitioning by Conditional Inference

```
1:  procedure RPCI(I, α, A, d)
2:  Input: I ⊆ (V ∪ ⊥)^|A| × E*, α ∈ [0,1] ⊂ ℝ, A ⊆ 𝒩, d ∈ A
3:  Output: O ⊆ ℙ((V ∪ ⊥)^|A| × E*)
4:      bestPValue ← 1                          ▷ worst possible p-value
5:      bestAttribute ← ⊥
6:      for all a ∈ A \ {d} do                  ▷ for all independent attributes
7:          pValue ← CORRELATIONTEST(I, a, d)   ▷ test correlation of a and d
8:          if pValue < bestPValue then
9:              bestPValue ← pValue
10:             bestAttribute ← a
11:         end if
12:     end for
13:     if bestPValue > α then
14:         return ({I})                        ▷ no significant correlation ⇒ no splitting
15:     else
16:         (I1, I2) ← SPLITWRTATTRIBUTE(I, bestAttribute)   ▷ I1 ⊆ I, I2 = I \ I1
17:         return RPCI(I1, α, A, d) ∪ RPCI(I2, α, A, d)     ▷ recursive step
18:     end if
19: end procedure
```

In the first part of the algorithm (lines 4 to 12 in Algorithm 1), the dependent variable d is tested for correlation with respect to each independent variable a in isolation using the method CORRELATIONTEST(I, a, d) (line 7 in Algorithm 1). From the independent variables for which the null hypothesis is rejected (i.e., they are significantly correlated to d), we select the one with the lowest p-value, which is obtained from the independence test. The lowest p-value (i.e., *bestPValue*) indicates the strongest significant correlation between the dependent variable and an independent variable. Note that if no null hypothesis is rejected (i.e., p-value larger than a given *alpha*), no splitting is done.

If an independent variable a is selected (i.e., a is significantly correlated to d), an *optimal binary partition* is searched (line 16 in Algorithm 1). A set of instances I can be partitioned using the independent attribute a in different

ways depending on the distribution of a.

For binary splitting a numerical a, a single value is chosen, which acts as a "border" between the resulting subsets. For example, splitting by the value 5 results in two groups of instances: $I_1 = \{\big((\#_a(Ev(\sigma)), \#_d(Ev(\sigma))\big), \sigma) \in I | \#_a(Ev(\sigma) \leq 5\}$ and $I_2 = \{\big((\#_a(Ev(\sigma)), \#_d(Ev(\sigma))\big), \sigma) \in I | \#_a(Ev(\sigma) > 5\}$.

For binary splitting a categorical a, a set of values is chosen, because categories are not comparable. For example, $I_1 = \{\big((\#_a(Ev(\sigma)), \#_d(Ev(\sigma))\big), \sigma) \in I | \#_a(Ev(\sigma)) \in \{Gold, Silver\}\}$ and $I_2 = \{\big((\#_a(Ev(\sigma)), \#_d(Ev(\sigma))\big), \sigma) \in I | \#_a(Ev(\sigma)) \notin \{Gold, Silver\}\}$. Note that RPCI provides several mechanisms to deal with missing values (i.e., $\perp$).

In the method SPLITWRTATTRIBUTE(I, a) (line 16 in Algorithm 1) RPCI uses the values of the independent variable a to detect a binary partition of the set of instances I into $I_1 \subset I$ and $I_2 = I \setminus I_1$ such that the difference between the distributions of the dependent variable d over the resulting subsets I_1 and I_2 is maximized.

In the last part of the algorithm (lines 14 and 17 in Algorithm 1), the whole process is repeated for the resulting partitions I_1 and for I_2 if a significant correlation between independent and dependent attributes was found. Otherwise, the input set of instances I is returned as the output.

As a result of the RPCI algorithm, the set of instances I is partitioned into non-overlapping (sub)sets of instances: RPCI$(I, \alpha, A, d) = \{I_1, ..., I_n\}$ where $I = I_1 \cup ... \cup I_n$. Given the recursive nature of this approach, the exact final number of obtained partitions depends on the characteristics and distributions of the variables, and how many independent variables are correlated to the dependent variable.

From Sets of Instances to Process Variants and Variant-Enriched Event Data

The last step of our approach consists of enriching the original event log with new event attributes that explicitly specify the process variants found in the event data.

Given an event log L and a set of instances I, each instance is denoted as $i = (a, \sigma) \in I$, where a is the sequence of values of the dependent and independent variables that describe the instance and $\sigma \in L$ is the trace related to such instance. The set of traces associated to I corresponds to a *process variant*, defined as $Var_I = \bigcup_{i=(a,\sigma)\in I}\{\sigma\}$

Then, for a set of instances I and a subset $I_m \subset I$, all the events in the process variant Var_{I_m} can be enriched with a new event attribute *variant* us-

ing a trace manipulation function defined as $\mathcal{T}_{VAR}(\langle e_1, \ldots, e_n \rangle) = \langle f_1, \ldots, f_n \rangle$ such that $\forall_{1 \leq i < n} : \#_{variant}(f_i) = m$ (see Section 2.2 for more details on trace manipulation functions).

This new "*variant*" event attribute encodes the specific variant to which the event is related, and can be used, e.g., as a dimension in a process cube for splitting an event log into such process variants.

6.3 Implementation

We have implemented our approach as a ProM [172] plugin named "Process Variant Finder" included in the *VariantFinder* package. We use the R library ctree [70] to perform the statistical tests and partitioning of the sets of instances, including the RPCI technique. Therefore, a running instance of R is required.[2] Figure 6.4 illustrates the flow of user interactions with our tool.

The first step in using our "Process Variant Finder" tool is to specify the settings that will be used to discover process variants, as shown in Figure 6.5.

In our tool, the "Settings" panel (Panel 1 in Figure 6.4) is sub-divided into three parts: "Transition System Settings", "Partitioning Settings", and "Actions", described as follows.

Transition System Settings: These settings define the transition system that will be used to identify points of interest (see Step 1 of our approach in Section 6.2). The user can also select a frequency threshold (i.e., "frequency" in Figure 6.5) that is used to filter out points of interest based on their relevance (see Definition 6.1).

Partitioning Settings: These settings are used to define instances that will be partitioned by RPCI, and also to parameterize the partitioning algorithm. This is done by defining the dependent variable (i.e., "class attribute" in Figure 6.5), the set of independent variables (i.e., "selected attributes" in Figure 6.5), the α for statistical tests (i.e., the statistical significance level), and also provides an early-stopping mechanism by defining the minimum size of a partition (i.e., "Min % of instances in a leaf" in Figure 6.5). Users can select whether to test all independent attributes in combination, or to test them separately (i.e., "use separately" checkbox in Figure 6.5).

Actions: This part contains two buttons: one for discovering process variants based on the specified settings defined above (i.e., "Apply Settings" in Fig-

[2] Our approach requires the R libraries "partykit" and "Rserve" to be installed an the function Rserve() needs to be executed. Such method opens a socket through which R communicates with Java.

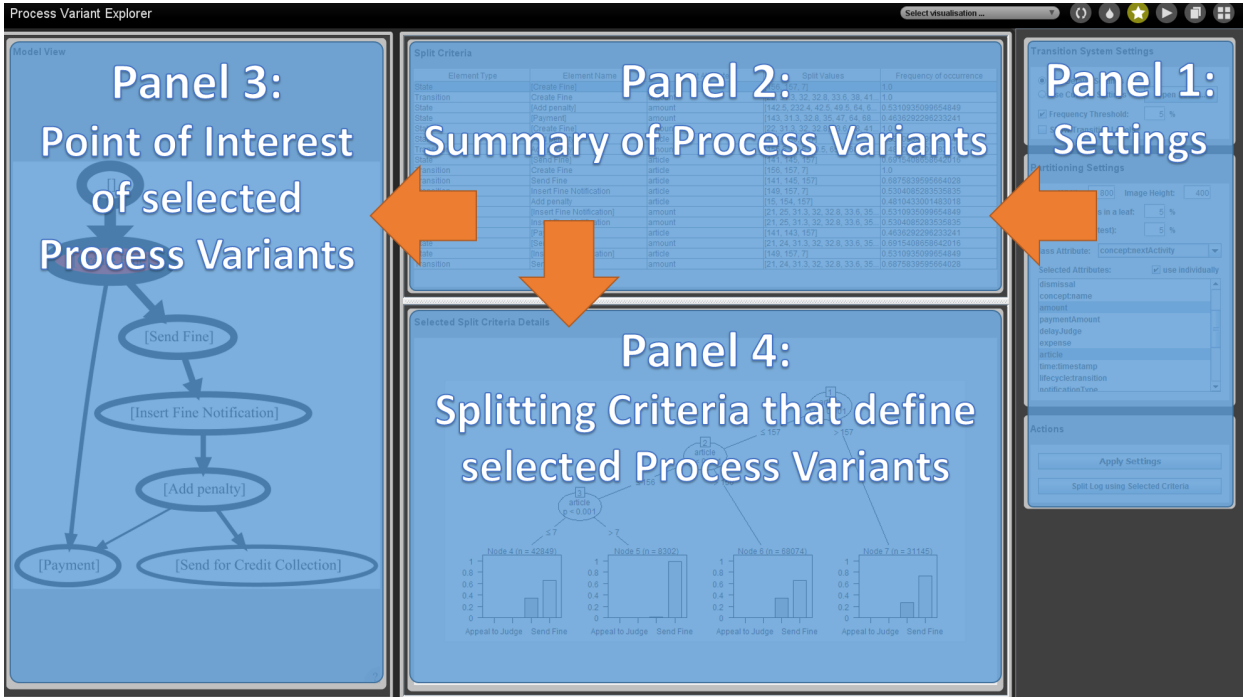


Figure 6.4: Illustration of the user flow for the "Process Variant Finder" tool. *Panel 1* shows the settings panel. Once the user clicks on the "Apply Settings" button, the tool searches for process variants in all the points of interest according to the specified settings. These results are shown in *Panel 2* which shows a table with information about the discovered process variants including: the type (i.e., state or transition), relevance (see Definition 6.1) and name of the points of interest where they were detected, and which independent attributes and values were used to split the event log into such process variants. When a row in *Panel 2* is selected (i.e., clicked) by the user, the corresponding point of interest is highlighted (in red) in the transition system that represents the process in *Panel 3* and the splitting criteria (attributes and values) that define such process variants is shown in *Panel 4*.

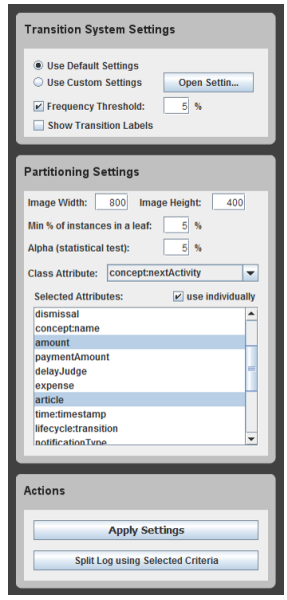


Figure 6.5: Settings panel or our tool (Panel 1 in Figure 6.4).

ure 6.5) and one that is used to enrich the event log with a new attribute specifying the corresponding process variants, according to the selection made by the user in Panel 2.

The details of the other panels (shown in Figure 6.4) and the step-by-step application of our tool to a real dataset and the interpretation of its results will be illustrated as follows.

6.4 Applications

This section first describes a step-by-step experiment showing the application of our technique to the event log related to the road fines management process (see Section 2.4). Then, Section 6.4.1 compares the obtained results with the arbitrary partitioning of this dataset that was performed in Chapter 5 (see Section 5.4.2).

The transition system used to represent the behavior in the road fines management process is shown in Figure 6.7 (based on Figure 2.6 presented in Sec-

tion 2.4). By default, all the states and transitions of this transition system are initially considered as point of interest, but only those with a relevance of 5% or more are used in our analysis.

In this experiment, we used the *next activity* attribute as dependent variable, and the attributes *amount* (i.e., the amount of the fine) or the *article* (i.e., the traffic law that was violated) were each used as independent variables (see Figure 6.5). Note that the independent variables were used separately. We used a significance level (α) of 5%.

After defining the settings, the second step in using our tool is to click on the "Apply Settings" button. Now, the "Summary of Process Variants" panel (Panel 2 in Figure 6.4) shows a table with a summary for all the process variants found, as shown in Figure 6.6.

Summary of Process Variants

Element Type	Element Name	Split Attribute	Split Values	Frequency of occurrence
State	[Create Fine]	article	[156, 157, 7]	1.0
Transition	Create Fine	amount	[22, 31.3, 32, 32.8, 33.6, 38, 41...	1.0
State	[Add penalty]	amount	[142.5, 232.4, 42.5, 49.5, 64, 6...	0.5310935099654849
State	[Payment]	amount	[143, 31.3, 32.8, 35, 47, 64, 68....	0.4636292296233241
State	[Create Fine]	amount	[22, 31.3, 32, 32.8, 33.6, 38, 41...	1.0
State	[Add penalty]	article	[141, 156, 157]	0.5310935099654849
Transition	Add penalty	amount	[155.5, 42.5, 49.5, 65.6, 68.77, ...	0.4810433001483018
State	[Send Fine]	article	[141, 145, 157]	0.6915408658642016
Transition	Create Fine	article	[156, 157, 7]	1.0
Transition	Send Fine	article	[141, 145, 157]	0.6875839595664028
Transition	Insert Fine Notification	article	[149, 157, 7]	0.5304085283535835
Transition	Add penalty	article	[15, 154, 157]	0.4810433001483018
State	[Insert Fine Notification]	amount	[21, 25, 31.3, 32, 32.8, 33.6, 35...	0.5310935099654849
Transition	Insert Fine Notification	amount	[21, 25, 31.3, 32, 32.8, 33.6, 35...	0.5304085283535835
State	[Payment]	article	[141, 143, 157]	0.4636292296233241
State	[Send Fine]	amount	[21, 24, 31.3, 32, 32.8, 33.6, 35...	0.6915408658642016
State	[Insert Fine Notification]	article	[149, 157, 7]	0.5310935099654849
Transition	Send Fine	amount	[21, 24, 31.3, 32, 32.8, 33.6, 35...	0.6875839595664028

Figure 6.6: Summary of Process Variants found in this experiment (Panel 2 in Figure 6.4). The *next activity* attribute was used as the dependent variable, and the attributes *amount* and *article* were used as independent variables.

Each row in this table relates to a set of process variants obtained from the event log using RPCI. Note that several sets of process variants can be found for the same point of interest depending on the independent attributes used for splitting (i.e., see the column "split attribute"). For example, rows 1 and 5 (highlighted in blue) relate to the point of interest defined by the state *Create Fine* (i.e., see the columns "split attribute"). However, in row 1, the process variants were obtained by splitting the event log based on the *article* independent variable (i.e., the traffic law that was violated) and in row 5 the process variants were obtained by splitting the event log based on the *amount* of the fine.

Once a set of process variants (i.e., a row in Figure 6.6) is selected by the user, Panels 3 and 4 in Figure 6.4 show more information about it. Note that the user can select only one set of process variants each time. In this demonstration, we select the *first row* of the summary of process variants which splits the event log into process variants based on the *article* of the fine in the point of interest defined by the state *Create Fine*. We chose such point of interest because it is the only one that is reached by all the traces in the event log.

As a consequence, the "Point of Interest of selected Process Variants" panel (i.e., Panel 3 in Figure 6.4) shows the location of the point of interest related to the selected set of process variants by highlighting it in red within the transition system that represents the behavior of the process. In this case, since the first row in the "Summary of Process Variants" panel was selected, the point of interest *Create Fine* is highlighted in red, as shown in Figure 6.7.

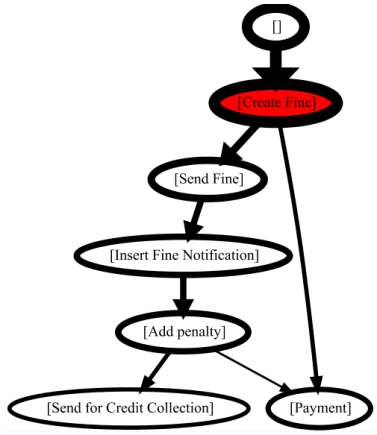


Figure 6.7: Point of Interest *Create Fine* of the set of Process Variants selected in this experiment (Panel 3 in Figure 6.4), with *next activity* selected as the dependent variable and *article* is selected as the independent variable.

At the same time, the "Splitting Criteria" panel (i.e., Panel 3 in Figure 6.4) describes the details of the splitting criteria used to partition the event log into the selected set of process variants by visualizing it as a tree where each *branch* of the tree defines the values of the independent variable that is used for splitting, and each *leaf* of the tree represents a *process variant*, and is visualized as the distribution of values of the dependent variable for the subset of instances that represent that variant. In this case, the process variants for the point of

interest *Create Fine* when only the *article* (i.e., traffic law that was violated) is used as an independent variable are shown in Figure 6.8. We can observe that

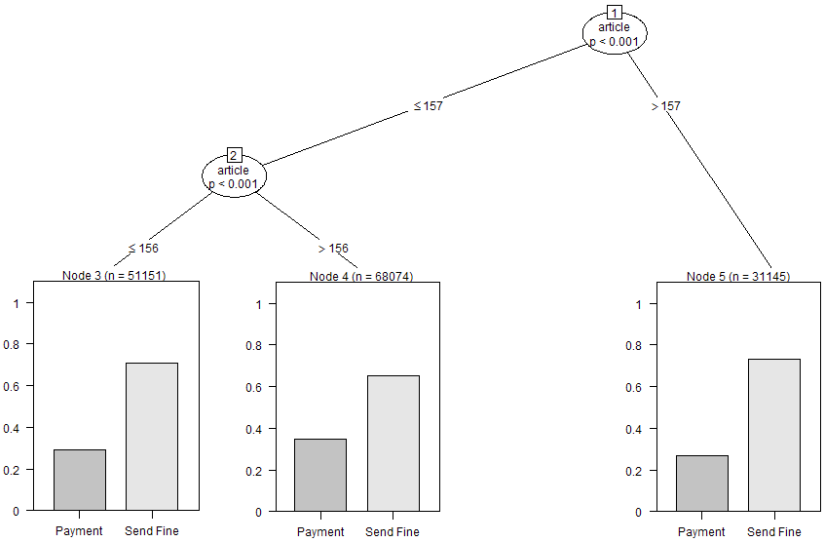


Figure 6.8: Process Variants found in the point of interest *Create Fine*. The dependent variable is the *next activity* to occur. The independent variable is the *article* i.e., traffic law that was violated (Panel 4 in Figure 6.4).

fines related to the *article* 157 (i.e., parking-related fines) have a significantly higher likelihood of being paid before the fine is sent to the offender, compared to other types of fines.

Figure 6.9 shows the process variants for the point of interest *Create Fine* when only the *amount* of the fine is used as independent variable. We can observe that more expensive fines (i.e., more than 41 euros) and cheaper fines (i.e., less than 32.8 euros) are less likely to be paid directly by the offender, and most likely the fine will be sent to the offender. We can also observe that fines with amounts between 32.8 and 41 euros are more likely to be paid directly (i.e., without the need of sending the fine by mail) by the offender than higher or lower-amount fines. We will use this observation to define process variants for the remainder of this section.

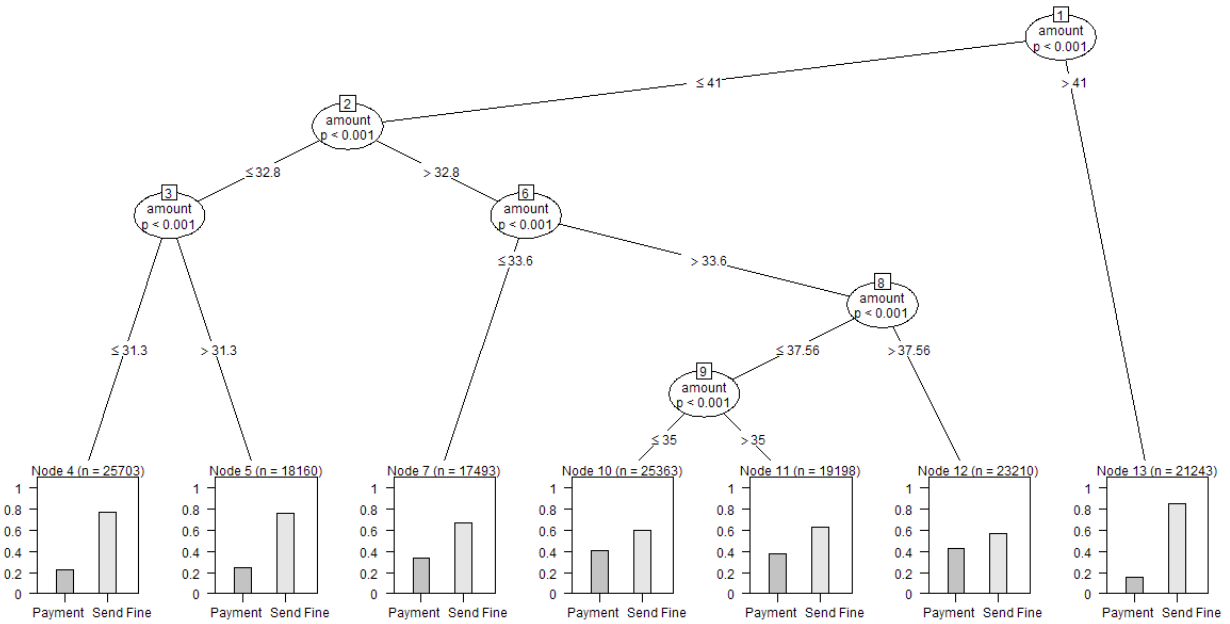


Figure 6.9: Process Variants found in the point of interest defined by the state `Create Fine`. The dependent variable is the *next activity* to occur. The independent variable is the *amount* of the fine. For each box, the X-axis represents the possible activity to be executed next, and the Y-axis represents the likelihood that an activity will be executed next.

6.4.1 Connection to Process Cubes and Comparison to Arbitrary Splitting of Data

In Section 5.4.2, we arbitrarily split this same dataset related to the read fines management process (see Section 2.4) into two process variants: one containing fines lower than 50 euro and the other containing fines higher than 50 euro, and then compared them for differences using the techniques described in Chapter 5. This type of arbitrary splitting is not uncommon in process mining analysis, as domain experts that could define a better splitting are not always available.

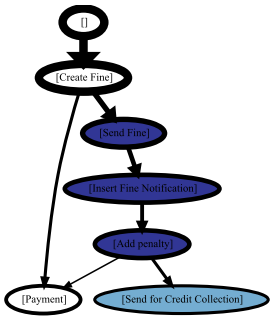
The purpose of the remainder of this chapter is to evaluate, in the whole process (not just in the state *Create Fine*), whether our approach provides a detection of process variants in which differences are more evident than on arbitrarily-chosen variants e.g., our previous 50-euro arbitrary split.

In the previous section chapter, we performed a different partition: traces with an amount between 32.8 and 41 euros define one variant (i.e., variant 1), and all the other fines define the other variant (i.e., variant 2). This choice, as previously discussed, is based on the results observed in Figure 6.9, where we grouped the variants that had a higher chance of being directly paid before the fine is sent. These variants are related to the point of interest defined by the state *Create Fine*. We used these results to enrich the event log by encoding the variants explicitly into the event data as a new *variant* event attribute (see Section 6.2.2). Then, we used a process cube (see Chapter 3) to split the event log by dicing the newly-created *variant* dimension so that the two process variants (i.e., 1 and 2) were contained in two different cells of the cube.

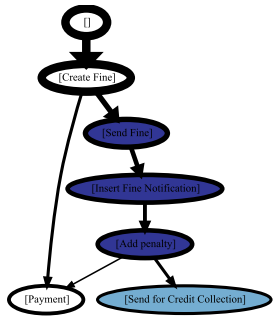
Given these two process variants, we used the technique described in Chapter 5 to compare them using the same three experiments discussed in Section 5.4.2: performance comparison, business rules comparison, and frequency of occurrence comparison.

In the first experiment (i.e., performance comparison), we compared these two variants in terms of their elapsed time. For more details about the design of this experiment, see Section 5.4.2. The results of the first experiment are shown in Figure 6.10.

We can observe that the two process variants found using our technique have much larger performance differences than just arbitrary splitting the event data. This can be observed in the fact that in the two process variants found using the technique described in this chapter, the darker blue colors of states *Send Fine*, *Insert Fine Notification* and *Add Penalty* indicate that the effect size is much larger compared to the same states if the event log is split like in Chapter 5. See



(a) Results of comparing *elapsed time* in fines higher than 50 euros and lower than 50 euros (extracted from Figure 5.15).

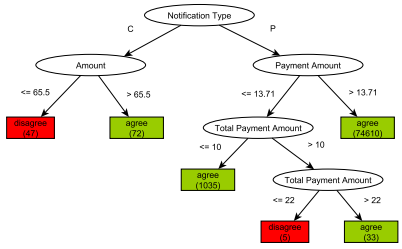


(b) Results of comparing *elapsed time* in fines between 32.8 and 41 euros and the rest of the fines.

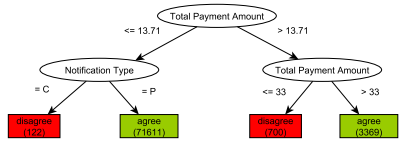
Figure 6.10: Performance (*elapsed time*) comparison between process variants.

Section 5.2.1 for more details about the color schemes used.

In the second experiment, we compared these two variants in terms of their business rules in the state *Add Penalty*. For more details about the design of this experiment, see Section 5.4.2. The results of the second experiment are shown in Figure 6.11.



(a) Agreement/Disagreement of business rules in fines higher than 50 euros and lower than 50 euros (extracted from Figure 5.19).



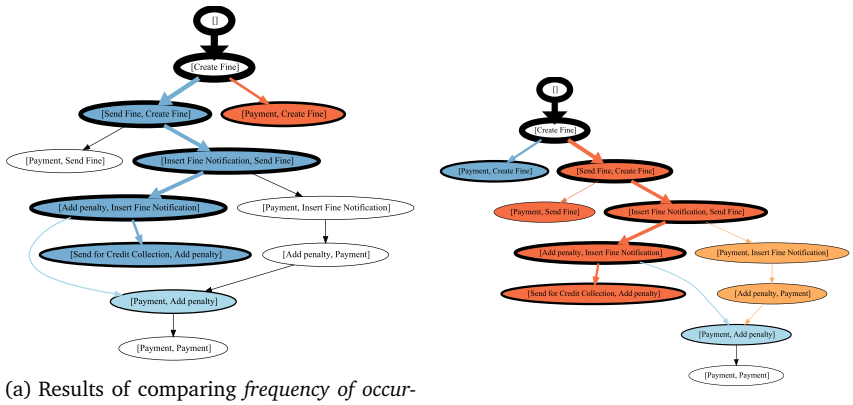
(b) Agreement/Disagreement of business rules in fines between 32.8 and 41 euros and the rest of the fines.

Figure 6.11: Business rules comparison between process variants.

We can observe that using the technique presented in this chapter, we could identify process variants that have larger differences in terms of business rules

than just arbitrary splitting the event data. This can be observed in the fact that in the two process variants found using the technique described in this chapter, we discovered that the business rules disagreed in 822 cases, compared to the 52 cases of disagreement if the event log is split like in Chapter 5.

In the third experiment (i.e., control-flow comparison), we compared these two variants in terms of their frequency of occurrence. For more details about the design of this experiment, see Section 5.4.2. The results of the third experiment are shown in Figure 6.12.



(a) Results of comparing *frequency of occurrence* in fines higher than 50 euros and lower than 50 euros (extracted from Figure 5.19).

(b) Results of comparing *frequency of occurrence* in fines between 32.8 and 41 euros and the rest of the fines.

Figure 6.12: Control-flow (*frequency of occurrence*) comparison between process variants.

We can observe that using the technique presented in this chapter, we could find process variants that have more significant control-flow differences. This can be observed in the fact that in the two process variants found using the technique described in this chapter, the color of the states *[Payment, Send Fine]*, *[Payment, Insert Fine Notification]* and *[Add Penalty, Payment]* indicate that the difference in terms of frequency of occurrence is significant, compared to the same states if the event log is split like in Chapter 5, where no significant differences were detected.

6.5 Conclusions

The problem of detecting process variants in event logs has been tackled by several authors in recent years. Many authors have successfully solved specific scenarios where the focus is put on specific attributes, such as time. Some have even provided general solutions, but they fail to filter out irrelevant splits. This chapter presents an approach that is able to detect relevant process variants in any process perspective (in the form of event attributes) by splitting any other (combination of) event attributes. The approach has been implemented and is publicly available. We also showed that the approach could rediscover designed performance issues in an artificially generated event log, where certain resources were related to poorer performance. We also were able to successfully identify points of process variability inside in a real-life event log and we were able to detect process variants without the use of domain knowledge, confirming such variability using process comparison techniques. Such process variants are used to enrich the original event log by explicitly encoding the process variants as event attributes. These variant-related attributes can be used as dimensions in a process cube, and can be used to split the event data, so that the resulting cells can be analyzed using process mining techniques, process mining workflows, or they can be compared using process comparison techniques. Therefore, our approach provides a viable solution to process variant detection, even when no domain knowledge is available.

As a limitation, we would like to mention that, even though our technique requires "less of it", it still relies on manual setting of parameters such as the independent and dependent variables and a few thresholds. Therefore, there is room for improvement in the sense of reducing the ammount of settings required to operate the tool by automatically choosing some of them.

Another limitation of the tool presented in this chapter is that it requires a connection to the R software to be functional. This can be difficult to set up for some people, and it is far from ideal. However, by the time of implementation, there were no libraries that implemented RPCI. The only place where RPCI was implemented was in R.

Part III

Large-Scale Experimentation

Chapter 7

A Framework for Benchmarking Process Discovery Techniques

Disclaimer: *The work presented in this chapter is based on the work presented in [85], in which the author collaborated with other researchers from TU/e and Hasselt University. This work was also included in the doctoral dissertation by Jouck [84], where the focus was put on the generation of process models and data. In this chapter, the focus is put on the evaluation capabilities of process discovery techniques, and on the large-scale experiments required for the statistical analysis of the results.*

The lion's share of attention within process mining was received by process discovery, which aims to discover a process model from event logs. This resulted in dozens of discovery algorithms (see [47, 156] for an overview). Researchers aim to improve the quality of the mined models to adequately represent the behavior observed in the event logs. A popular way to measure the quality is the *fitness* between the event log and the mined model. However, there are other ways to measure quality as well (e.g., precision, generalization, simplicity). A good model allows for the behavior seen in the event log. Fitness alone is not sufficient, also a proper balance between *overfitting* and *underfitting* is required [156]. A process model is *overfitting* (the event log) if it is too restrictive, disallowing behavior which is part of the underlying process but not yet

observed. This typically occurs when the model only allows for the behavior recorded in the event log. Conversely, it is *underfitting* (the reality) if it is not restrictive enough, allowing behavior which is unlikely to be part of the underlying process. This typically occurs if it overgeneralizes the observed behavior in the event log.

The abundance of discovery algorithms has made it increasingly important to develop evaluation frameworks that can compare the efficiency of these discovery techniques, especially in terms of balancing between overfitting and underfitting. As detailed in Section 7.1, several comparison frameworks have already been proposed in literature. Unfortunately, these frameworks are characterized by at least one of the following three major limitations:

1. They are not independent from the modeling notation in which the discovered models are represented, e.g. two behaviorally-equivalent models may have very different precision scores, or quality can only be measured after a conversion that does not preserve the behavior precisely. This restricts the framework to a comparison of the algorithms that generate models in one specific notation.

2. The evaluation results are based on concrete event logs and cannot be generalized as the population of processes from which they originate is unknown. Processes come from different populations depending on the type of behavior allowed. Processes may have different behavioral characteristics, with parts that can be repeated, with mutually-exclusive and parallel branches, with non-local dependencies and so on. Also, these characteristics can be more or less predominant in a process model. Different algorithms may better deal with a certain characteristic than others. And the quality of the discovered model may also depend on the predominance of certain characteristics. Performing a comparison without acknowledging the influence of these behavioral characteristics can lead to inconclusive results.

3. They use a small number of processes. Even if processes are randomly sampled from a well-defined population of processes, one cannot validate the evaluation results if only a few processes were considered, as this prevents the results from being statistically and generally valid.

This chapter tries to overcome these limitations by proposing a framework that:

1. abstracts from the modeling notation employed.

2. starts from the definition of a process population where the probability of several behavioral characteristics can be varying. From this population a random sample of process models and event logs is drawn, thus making it possible to evaluate and generalize the influence of behavioral characteristics on the quality of the discovered models by the different algorithms under analysis.

3. performs experiments on random samples of a user-specified size, so as to return statistically valid results.

In a nutshell, our framework is based on a classification perspective to evaluate the quality of a discovered model. The framework starts with artificially generating random samples of process models from a specified population of processes. For each model, we generate a training log with fitting traces (to discover a model) and a test log with both fitting and non-fitting traces (to check conformance). Then, the quality of a discovery algorithm with respect to the event log is related to the ability to correctly classify the traces in the test event log: the discovered model should classify a trace representing real process behavior as fitting and a trace representing a behavior not related to the process as non-fitting. In this way the classification approach allows us to evaluate discovery algorithms generating models in different modeling notations because the quality measurement is not based on one specific modeling notation. Furthermore, by using (large) samples of randomly generated models and logs we can make general statements about populations of models and logs.

Obviously, repeating the generation of event logs and process models cannot be done manually to get significant results. We aim at thousands of models and logs in order to generalize. Fortunately, this can be automated through the use of process mining workflows, presented in Chapter 4.

In summary, this chapter introduces a novel evaluation framework which is operationalized using a process minig workflow, and is an instantiation of the use case "*Large-scale experiments*" proposed in Section 4.4.3. The experiments report the results of their application to five state-of-the-art discovery algorithms. It is beyond the scope of this thesis to extensively cover every existing discovery algorithm. However, the operationalization and the experiments show how easy it is to extend the framework to other algorithms.

The remainder of this chapter is structured as follows. Section 7.1 discusses related work. Section 7.2 discusses the new evaluation framework including the methodological foundations and the building blocks used. Next, Section 7.3 describes the implementation of this framework as a process mining workflow

and its application in an experimental setting, providing a discussion of the experimental results. Finally, Section 7.4 concludes the chapter.

7.1 Related work

Several frameworks for evaluating process discovery algorithms have been proposed. Rozinat et al. [130] introduced the first evaluation framework, Wang et al. [185] and Ribeiro et al. [126, 127] extended the Rozinat framework to evaluate and predict the best algorithm. Weber et al. [187] proposed an alternative framework that takes a probabilistic perspective. In addition to the evaluation frameworks, De Weerdt et al. [46, 47], Vanden Broucke et al. [177, 179] and Augusto et al. [6] performed benchmarking studies of process discovery techniques.

As indicated before, our framework evaluates the quality of models on the basis of measures of *precision* and *recall* that are not bound to any modeling notation. Conversely, the existing body of research is based on metrics that are applicable to one notation, mostly Petri nets [6, 47, 126, 127, 130, 177, 179, 185, 187].

We previously mentioned that the second and third advantage of our framework is that it is based on the generation of a sufficiently-large number of artificial models to guarantee a statistical validity of the analysis. Conversely, existing frameworks base their conclusions on samples that are small, either a few real-life event logs [6, 46, 47, 127], either artificial but not randomly generated [47, 126, 127, 130, 179, 185, 187], thus limiting the statistical validity of the analysis. Also, the artificial process models are not generated by controlling the probability of certain constructs to be present. This means that the event logs generated from these models do not allow one to evaluate the correlation between the quality of the discovered models and the presence of certain process constructs.

Furthermore, all frameworks, except Weber et al. [187], leverage on the typical process-mining notions of precision, generalization, and fitness from literature to evaluate the quality of the discovered models (see, e.g., [156]). The work presented by van der Aalst [168] discusses an exhaustive taxonomy of 21 propositions for conformance measures. Tax et al. [146] introduces a discussion on the quality of precision measures in process mining. All these process-mining measures are designed considering that the *real process model is not known* and that one only observes the positive cases, namely the traces that are part of the real process. The negative cases (i.e., the executions (traces) that do not fit the

real process) are not known because they would require to know the real model. Therefore, the process-mining measures of model quality try to artificially generate the negative cases based on estimation and, hence, the measured results are estimates. An example of this is the work presented in [61, 178], where negative events are added to event logs. Negative events record that at a given position in a trace, a particular event cannot occur. However, negative events are created based on the lack of evidence that contradicts them. This means that negative events do not necessarily represent behavior that is not allowed by the real process model: it only represents behavior that has not been observed yet. In our approach, we *know the real underlying process*: we know what behavior is allowed and not allowed by the real process model, and we also know which behavior has been observed and which behavior has not been observed yet. Therefore, the precision and recall measures that we use in our approach are closer to the ground-truth than the typical process-mining measures of model quality.

The frameworks reported in [87, 88] are clearly not the only to generate process models and event logs. While the framework would allow one to plug different model and log generators, the choice has fallen onto those frameworks because they provide an API that allows one to invoke them from code, as our scientific workflow requires. For example, PLG [30] only allows a GUI interaction; also, PLG does not support certain patterns, namely long-term dependencies, silent transitions, and duplicate activity labels.

The classification approach of the proposed evaluation framework builds upon established principles and methods from the machine learning domain. See [78] for more information on the empirical evaluation of learning algorithms using a classification perspective.

7.2 Discovery Evaluation Framework

The framework presented in this chapter aims to evaluate the quality of discovery algorithms to rediscover a model when confronted with a fraction of its behavior. The framework is designed based on the principles of scientific workflows and experimental design. The former captures the complete evaluation experiment in a workflow that can be automated, reused, refined and shared with other researchers [8]. The latter allows for precise answers that a researcher seeks to answer with the evaluation experiment [93].

To integrate the steps needed for empirically evaluating process discovery algorithms, the framework is built as a *process mining workflow* (see Chapter 4).

Process mining workflows offer several advantages over traditional ways to conduct process discovery evaluation. The first advantage comes from workflow automation. Experiments evaluating discovery techniques involve large-scale and computationally expensive experiments that require intensive human assistance. Therefore, automating these experiments removes the need for the human assistance and reduces the time needed to perform experiments. A second benefit comes from the modularity of the workflows. This allows researchers to adapt and extend an existing workflow, e.g., by using other parameter settings or adding new process discovery techniques. A final benefit is that they can be shared with other researchers. As a result other researchers can replicate experiments with little effort. In this way, our framework facilitates repeated process discovery evaluation, e.g. it becomes trivial to evaluate another set of algorithms or to assess the algorithm's performance with regard to other data characteristics (e.g. noise, control-flow patterns, etc.).

An evaluation analysis aims to test statistical hypotheses about a discovery algorithm. For example, does the presence of loops cause the Alpha+ miner [167] to discover models with lower fitness? Or: do the Alpha+ miner and Heuristics miner [189] perform equally in the fitness dimension on event logs with *non-exclusive choice (OR)* behavior? This makes it fit within the experimental design methodology in which the primary goal is to establish a causal connection between the independent (algorithm, log characteristics) and dependent (model quality criteria) variables [93]. The three cornerstones of good experimental design are: *randomization*, *replication*, and *blocking* [55]. The three cornerstones together are fundamental to make the experiments scientifically sound (e.g., avoid bias or wrong conclusions). Therefore, the evaluation framework incorporates each of the cornerstones.

Randomization involves the random assignment of subjects to the treatment in order to limit bias in the outcome of the experiment [48, 93]. In the evaluation context, the subjects are the event logs and the treatments are the discovery algorithms. Therefore, the evaluation framework has to ensure that the event logs generated in the data generation step are random observations from a population of processes with all desired control-flow characteristics.

Replication means that more than one experimental unit is observed under the same conditions. It enables researchers to estimate error effects and obtain a more precise estimate of treatment effects [93]. In the context of process discovery this implies that one needs to test a specific algorithm on more than one event log drawn from the same population to accurately assess the effect of that algorithm on model quality. The framework requires that the evaluation is based on a sample of event logs from a given population to obtain better

estimates of the studied effect.

Finally, blocking an experiment is dividing the observations into similar groups. In this way one can compare the variation between groups more precisely [48]. For example, if the experiment studies the effect of loops on model quality, also other characteristics such as infrequent behavior could have an effect. Therefore, the evaluation framework allows to vary the presence of loops in models (variable of interest) while holding the infrequent behavior constant to obtain precise estimates of the effect of loops on model quality (studied effect).

The remainder of this section is organized as follows. Section 7.2.1 describes the design of the framework. Section 7.2.2 describes the building blocks used in this framework in more detail. Finally, Section 7.2.3 discusses the extensibility of the framework.

7.2.1 The Design and Use of the Evaluation Framework

The framework focuses on evaluating control-flow discovery algorithms. Therefore, other process related perspectives, such as data and resources, are out of scope. Moreover, the framework aims at evaluation instead of predicting the best performing algorithm given an event log. The framework enables two main objectives: either benchmarking different discovery algorithms, or performing sensitivity analysis e.g., what effect does a control-flow characteristic or event log characteristic have on the algorithm's performance.

Figure 7.1 illustrates the framework instantiating the "large-scale experiment" use case (see Section 4.4.3). The framework uses the *process mining building blocks* defined in Section 4.2, represented as grey boxes. It also contains non-process-specific blocks, represented as white boxes. Such blocks represent generic data-processing functionalities (e.g., statistical testing) that complement the process mining building blocks in order to obtain the desired analysis results.

The framework enforces the consecutive execution of data generation, process discovery, quality measurement and statistical analysis. The framework applies a classification approach to allow for the evaluation of discovery algorithms generating models in different notations.

The first step i.e., the data generation, is triggered by the objective of the experiment. As a result, the objective determines the control-flow behavior a researcher wants to include in the event logs. The specification of control-flow behavior defines a population of process models. This population definition is the start of the data generation phase.

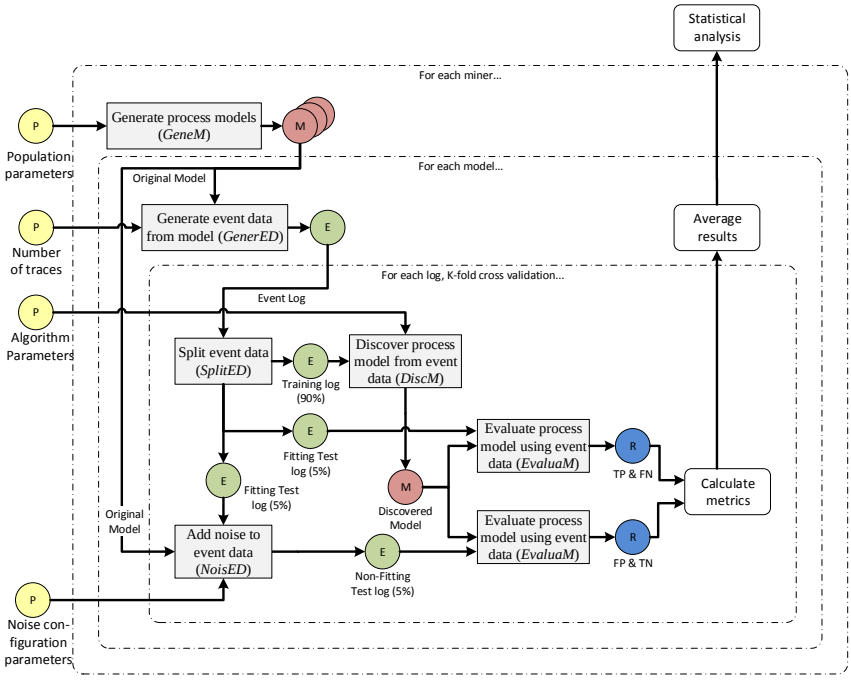


Figure 7.1: Framework for process discovery algorithm evaluation, presented as a process mining workflow. Grey boxes represent process mining building blocks. White boxes represent non-process-mining operators.

For each discovery algorithm to be tested, multiple instances of the "generate process models" block run in parallel. The generation results in multiple *random* samples of process models from the same population. Each model ("original model") is then simulated once by the block "generate event data from model" to create one event log i.e., a *random* sample of traces from all possible traces allowed by the model. The samples of process models and event logs constitute as *the ground truth*.

Next, the k-fold cross-validation splits each event log into k subsets (i.e., folds) of equal size. $k-1$ folds form the training log, while the remaining fold serves as the test log.

The block "discover process model from event log" applies the algorithm to discover a model from the training log.

To this point, the test log only contains positive examples i.e., traces that fit the original model. The classification approach requires also negative examples i.e., traces that do not fit the original model. To generate negative examples, the block "add noise to event data" alters half of the test traces until they cannot be replayed[1] anymore by the original model (i.e., *the ground truth*) to create non-fitting traces. Thus, the test log contains a 50/50 balance between fitting and non-fitting traces to avoid the class imbalance problem which makes the evaluation more difficult [79].

Subsequently, the framework measures the quality of the discovery algorithm by using the discovered model to classify the test traces. This classification happens within the "evaluate process model using event data" block which replays all traces on the discovered model (i.e., conformance checking). A trace representing real process behavior should be classified as allowed i.e., completely replayable. A trace representing behavior not related to the real process should be classified as disallowed by the discovered model i.e., not completely replayable. This approach allows for evaluating any discovery algorithm generating models with formal replay semantics.

The classification results are then combined in a confusion matrix in the block "calculate metrics". This is discussed in the next section. Based on that matrix, one can compute the well-known *recall* and *precision* metrics to evaluate the quality of the discovery algorithm.

The framework repeats the process of splitting, discovery, creating non-fitting traces and conformance checking ten times, each time with a different fold as the "test log". The block "average results" computes the average of the metric values over the ten folds to get an estimate of the algorithm's performance. Note that by using k-fold cross validation the obtained estimate is less likely to suffer from bias i.e., it helps to decrease the difference of the estimate from the real unknown value of the algorithm's performance on the population of processes. Finally, the block "statistical analysis" tests the hypotheses formulated in the context of the objectives.

This framework's design has the property that no two discovery algorithms are applied to the same event log. Furthermore, for each generated model - randomly drawn from a predefined population of models - we randomly draw only a single event log. Consequently, all discovered models and all corresponding

[1] Replay uses the trace and the model as the input. The trace is "replayed" on top of the model to see if there are discrepancies between the trace and the model [156].

quality metrics are independent observations which is an important assumption made by many standard statistical techniques. We acknowledge that this design decision is not the only option, as one could test discovery algorithms on the same logs. This alternative design would have more statistical power for the same sample size, however, it requires more complex statistical techniques to deal with the dependence between observations. We can compensate for the loss in power in our design by defining the desired power in advance and calculate the sample size required for such power.

Finally, the framework's design based on the experimental design principles enable users to obtain algorithm's performance measures that are *independent* from specific process models and event logs. More specifically, this starts from the generation of (preferably large[2]) random samples of process models and logs from a population, which are then used to estimate the performance of an algorithm with regard to that population. This contrasts evaluations based on a small non-random sample of (manually created) process models and event logs as it could influence the performance estimate to only reflect these particular models and logs.

7.2.2 The Building Blocks of the Framework

The previous section described the design of the framework, and briefly described the blocks that conform it. This section elaborates on each of the blocks presented above.

Generate process models

This building block (i.e., *GeneM*, introduced in Section 4.2.3) generates a random sample of process models from a population of models. The input of this block is a set of parameters describing the population characteristics. The user can specify such population characteristics by assigning probabilities to control-flow characteristics of the model (e.g., parallelism) and setting the size of the models in terms of visible activities. The probabilities of the control-flow characteristics influence the probability for each characteristic to be included in the resulting process model. For example, if the probability of loops is 0.2, then on average 20% of the model control-flow constructs will be of type loop.

[2]One can define the required sample size based on the desired power of the statistical analysis in advance, see [38].

There are several approaches in literature that can be used to generate process models, e.g., PLG [30,32], GraphGrammar [90], BeehiveZ [83], TestBed [82], and PTAndLogGenerator [88]. All these approaches are able to generate random samples of process models from a population of models, and can implement this building block.

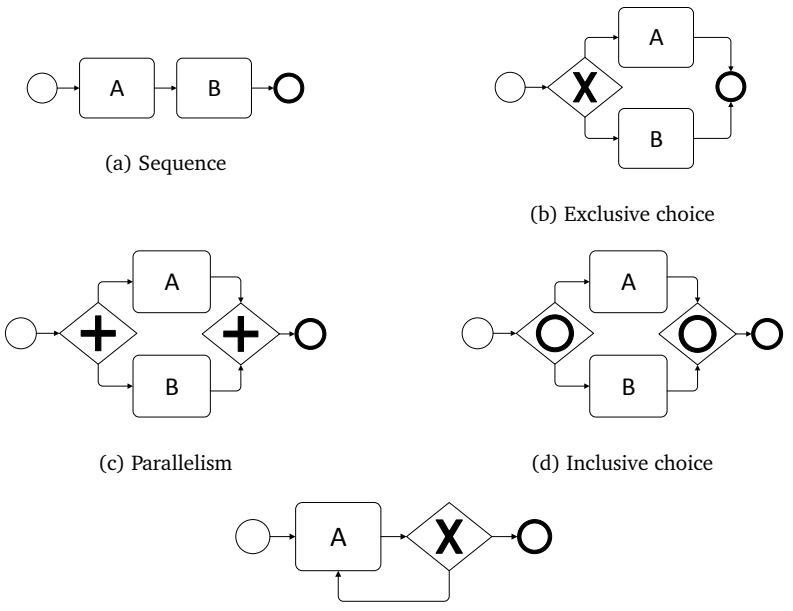
In this thesis, we chose to use the PTAndLogGenerator [88] approach given its wider range of parameters to specify control-flow characteristics. As a few examples: PLG and TestBed cannot specify the size (i.e., the number of activities) of a model, GraphGrammar and BeehiveZ cannot create models with loops, and none except PTAndLogGenerator can generate models with OR constructs, or with duplicate activities. See [88] for an exhaustive comparison of these approaches.

The approach used in this thesis as the implementation of this building block allows one to generate models in the form of *process trees* (described in Section 2.3) with random nestings of several basic control-flow patterns identified in [132]. These basic control-flow patterns are explicitly supported by most process modeling notations e.g., BPMN, Petri nets, process trees. Figure 7.2 illustrates these patterns through examples in BPMN notation.

The *sequence* pattern is illustrated in Figure 7.2a: first, the activity A is executed, then the activity B is executed. The *exclusive choice* (i.e., XOR) pattern is illustrated in Figure 7.2b: either A or B is executed. The *parallelism* (i.e., AND) pattern is illustrated in Figure 7.2c: both activities A and B are executed in no particular order. The *inclusive choice* (i.e., OR) pattern is illustrated in Figure 7.2d: either A or B, or both A and B are executed (in no particular order). Finally, the *loop* pattern is illustrated in Figure 7.2e: activity A can be executed multiple times.

This set of patterns is complemented by a set of more advanced control-flow patterns, illustrated in Figure 7.3 as examples in BPMN notation.

The *silent transitions* pattern is illustrated in Figure 7.3a: a "silent transition" (i.e., unlabeled activity, highlighted in black) is inserted so that activity A can be skipped. The *duplicate activities* pattern is illustrated in Figure 7.3b: activity A is present in multiple parts of the process. The *long-term dependency* pattern is illustrated in Figure 7.3c: the choice between activities C and D is affected by the previous choice of A and B i.e., C can be chosen only if A was chosen before. Finally, the *infrequent paths* pattern is illustrated in Figure 7.3d: when the execution reaches an exclusive choice, one outgoing branch (i.e., activity A) has a 90% chance to be chosen, while all other branches (i.e., activity B) have a combined 10% chance to be chosen. In the absence of infrequent paths, all branches have the same probability of being chosen. Note that this pattern



(a) Sequence

(b) Exclusive choice

(c) Parallelism

(d) Inclusive choice

(e) Loop

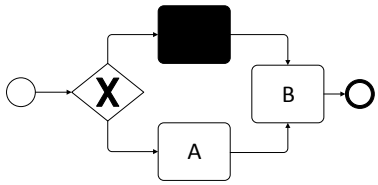
Figure 7.2: Basic control-flow patterns.

impacts the frequency of traces in the event logs, as traces will tend to follow the "frequent paths". Also note that the frequency of an "infrequent path" is determined by the number of branches in the exclusive choice.

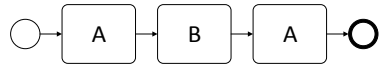
As a result, the implementation of this block allows users to fully control the control-flow behavior in the generated models and generalize the results to the pre-defined population. Concretely, the user can define a population of process models by setting a value for each of the following parameters related to the patterns described above:

Generate event data from model

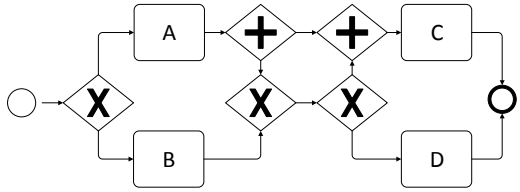
For each generated model, this block creates an event log i.e., a random sample of all possible traces allowed by that model. This building block simulates the given model to generate a user-specified number of traces per event log. As



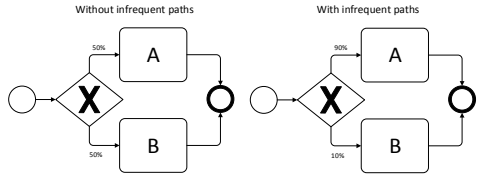
(a) Silent transitions



(b) Duplicate activities



(c) Long-term dependency



(d) Infrequent Paths

Figure 7.3: Advanced control-flow patterns.

a result, the resulting event log contains a random set of fitting and complete traces.

The exclusive choices in each of the models have output-branch probabilities. Therefore, the presence of infrequent paths will affect these probabilities in order to make some branches of the model more visited than others, which will result in event logs with infrequent behavior. The technique used in this framework for generating event data is described in [88].

Table 7.1: Parameters used to define a population of process models.

Parameter Type	Parameter	Value set	Constraints
Model size (# act)	min	$\mathbb{N}$	min $\leq$ mode $\leq$ max
	mode	$\mathbb{N}$	
	max	$\mathbb{N}$	
Probability of basic c-f pattern	sequence (s)	$[0, 1] \subset \mathbb{R}$	$s + e + p + i + l = 1$
	exclusive choice (e)	$[0, 1] \subset \mathbb{R}$	
	parallelism (p)	$[0, 1] \subset \mathbb{R}$	
	inclusive choice (i)	$[0, 1] \subset \mathbb{R}$	
	loop (l)	$[0, 1] \subset \mathbb{R}$	
Probability of advanced c-f pattern	silent transitions	$[0, 1] \subset \mathbb{R}$	none
	duplicate activities	$[0, 1] \subset \mathbb{R}$	
	long-term dependency	$[0, 1] \subset \mathbb{R}$	
	infrequent paths	{true,false}	

Split event data

This building block applies the first step needed for the k-fold cross validation evaluation method. The step splits a given event log into k subsets (folds) of equal size. $k - 1$ folds will form the "training log" and are the input of the discovery algorithm. The k-th fold is the "test log" which is split in half: one half constitutes the "fitting test traces", the other half will serve as the input of the "add noise to event data" block to make "non-fitting test traces". This is repeated k times such that each of the k folds becomes a "test log" exactly once.

Add noise to event data

In a classification approach, the "test log" should contain positive and negative examples. To this point, there are only positive examples i.e., an event log containing only traces that fit the original model. The "add noise to event data" block alters the given test traces so that they do not fit the original model anymore. The goal of the non-fitting traces is to punish overgeneralization of discovery algorithms. The flower model is an example of extreme overgeneralization that allows every possible trace but provides no added value in a business context [156]. Therefore, this block aims to punish typical overgeneralizing patterns such as unnecessary loops, activity skips and parallelism, by altering the traces using specific *noise operations* (see description below) that can add or remove behavior. Additionally, the traces are altered but kept as

close to the original trace as possible. In this way, the framework avoids non-fitting traces that would be trivially rejected by underfitting models e.g., adding an activity name that is not present in the original model.

It is important to note that the generation of non-fitting traces is done in consideration of both the event log and the original model, as shown in Figure 7.4. We consider three types of traces: non-fitting traces i.e., NFT, represent behav-

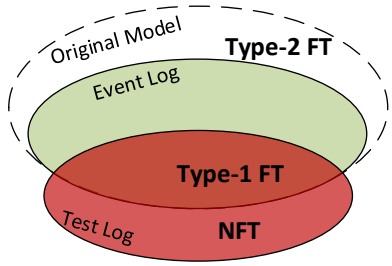


Figure 7.4: Illustration of a test log composition. Non-fitting traces (i.e., NFT) do not fit the original model. Type-1 fitting traces fit the original model, and have been observed in the event log. Type-2 fitting traces fit the original model, but have not been observed in the event log. A test log is composed of type-1 fitting traces and non-fitting traces.

ior that is not considered by the original model. Type-1 fitting traces represent the behavior observed in the event log and considered in the original model. Type-2 fitting traces represent the behavior considered in the original model, but not observed in the event log. Note that both non-fitting traces and type-2 fitting traces have not been observed in the event log. As mentioned earlier, most existing approaches for evaluating discovery techniques do not consider the original model, and use only the event log instead. If non-fitting traces are generated only considering the event log (hence, disregarding the original model), then such traces could be either *real* non-fitting traces or type-2 fitting traces. In this chapter, we use both the event log and the original model to generate non-fitting traces and overcome this limitation. Therefore, in the test logs used in this chapter, only type-1 fitting traces and non-fitting traces are included.

Given a process model and a set of type-1 fitting traces, noise is randomly-added to each trace in order to transform it into a non-fitting trace as follows.

First, one or more of the following noise types based on [58] is added with a user specified probability:

- *Add activity*: one of the process activities is added in a random position within the trace.
- *Duplicate an activity*: when an activity is duplicated, it is inserted immediately after the original.
- *Remove an activity*: a single activity is randomly removed from the trace.
- *Swap consecutive activities*: a random pair of consecutive activities are swapped within the trace.
- *Swap random activities*: similar to the previous type of noise, but the activities to be swapped are selected from random positions in the trace (not necessarily consecutive).

Then, the modified trace is checked for fitness with respect to the original model. If the trace does not fit anymore, it is a noisy trace which will not be edited anymore. If the trace still fits the model, noise is added again (and checked afterwards) until it does not fit anymore, or until noise has been added five times. If the noisy trace still fits the model, the trace is discarded and another trace is randomly selected from the set of fitting traces. This trace follows the same process described above.

Discover process model from event data

This block applies a discovery algorithm to the "training log" to induce a process model. This could be any discovery technique with user specified parameter settings. The discovered model will be used for conformance checking in the next block.

Evaluate process model using event data

This block consists of a conformance check: the given traces used as input are "replayed" on the discovered model. Because the framework applies a classification approach, the replay assigns each trace to a binary class: if a trace can be completely replayed by the discovered model it belongs to the "fitting" class, otherwise the trace is a part of the "non-fitting" class. The number of classes could be extended to create a more fine-grained evaluation. However, we argue

that determining the classes for partially fitting traces would require additional research, which is outside the scope of this thesis.

Calculate metrics

The framework summarizes the performance of an algorithm using three standard metrics adopted from the data mining and information retrieval domain: precision, recall and F-measure. Traditionally these metrics are based on:

- True Positives: the number of real traces that **fit** the discovered model.
- False Positives: the number of false traces that **fit** the discovered model.
- False Negatives: the number of real traces that **do not fit** the discovered model.
- True Negatives: the number of false traces that **do not fit** the discovered model.

The *precision* metric refers to the percentage of traces that fit the **original** model from all the traces that fit the **discovered** model.

$$\text{Precision} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Positives})}$$

Inversely, the *recall* metric refers to the percentage of traces that fit the **discovered** model from all the traces that fit the **original** model.

$$\text{Recall} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Negatives})}$$

The framework uses the F_1 variation of the F-measure. This statistic refers to the harmonic average of the precision and recall metrics.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{(\text{Precision} + \text{Recall})}$$

Statistical analysis

The evaluation framework allows users to compare the performance of algorithms and to study the effect of control-flow characteristics on the performance of the algorithm. The statistical analysis based on the evaluation results depends on the objectives of the experiment and the corresponding hypotheses

to be tested. Therefore, the framework does not incorporate specific statistical techniques, instead it can be used with a whole range of exploratory, descriptive, and causal statistical techniques to test any hypothesis that can be expressed in terms of precision, recall, F_1 score, and characteristics of log and model. We believe that this will benefit the adoption of the framework for all types of evaluation studies, rather than serve a specific purpose.

7.2.3 Extensibility of the Framework

As claimed before, the framework reported in this chapter is not bound to Petri nets or any other modeling notation. Concretely, we included Petri-net-based and declarative-model-based approaches in the benchmark. As a consequence, it is extensible to incorporate new discovery algorithms, independently of the notations in which these algorithms generate the model. Note that in such cases, the framework workflow (illustrated in Figure 7.1) is not affected because the corresponding blocks are not bound to a specific implementation.

To add a new algorithm to the implementation of the framework, it is necessary to (1) plug-in the new algorithm as a new instantiation of block *Discover process model from event data* (discussed in Section 7.2.2) and (2) plug-in a new conformance checker, if needed, for the notation used by the discovery algorithm. Note that it is not necessary to change the instantiation of block *Generate process models*. Any model generator in any notation that can represent the patterns defined in Section 7.2.2, such as process trees, can be employed. These models are only used to generate the event logs with fitting and non-fitting traces and are not directly compared with the models that are discovered.

For example, consider the case that one wants to evaluate an algorithm that discovers a BPMN model while limiting the number of changes to the current implementation. The implementation of the algorithm needs to be plugged into RapidProM. Also, a conformance checker for BPMN models needs to be available in the implementation. As a matter of fact, this conformance checker is already available in the implementation. First, the BPMN model is converted into a Petri net that is trace equivalent: each execution of the BPMN is possible in the Petri net, and vice versa [89]. Second, the Petri-net conformance checker can be employed. The trace equivalence between the BPMN and the Petri net models guarantees that every trace that is diagnosed as fitting/unfitting using the equivalent Petri net will also be as such with respect to the original BPMN model.

7.3 Experiments

The framework presented in this chapter has been implemented as a *process mining workflow* (see Chapter 4). Therefore, other researchers can replicate experiments with little effort by just executing the workflow. In this way, our framework facilitates repeated process discovery evaluation, e.g. it becomes trivial to evaluate another set of algorithms or to assess the algorithm's performance with regard to other data characteristics (e.g. noise, control-flow patterns, etc.).

The framework was operationalized in RapidProM (presented in Chapter 4), which contains instantiations for all the blocks mentioned in Section 7.2.2. Figure 7.5 illustrates the implemented workflow.

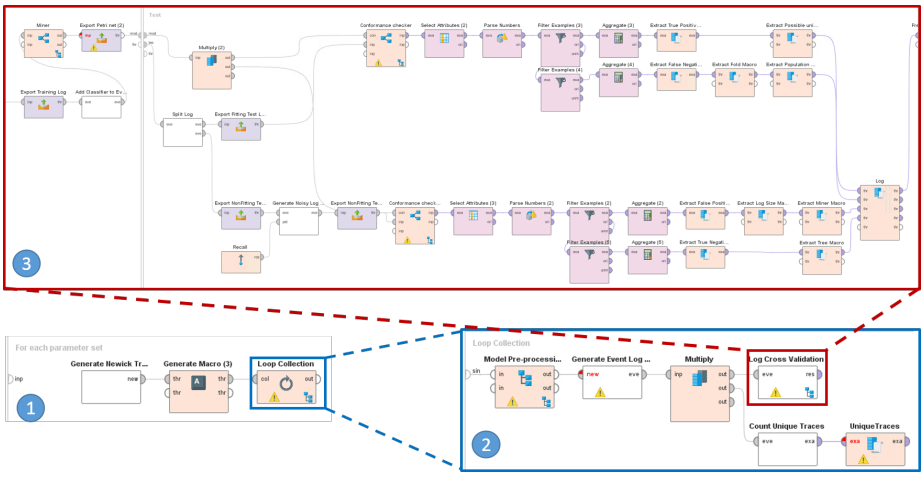


Figure 7.5: Concrete implementation of the framework into a RapidMiner workflow. In Step 1, a collection of models is generated from a population settings parameter. In Step 2, for each generated model, an event log is created. In Step 3, each event log is used to rediscover a model using different miners (left side), which are checked for conformance with respect to fitting and non-fitting traces (right side). Finally, results are processed.

The experiments were based on the following discovery algorithms: Alpha+ miner [167], Heuristics miner [189], ILP miner [170], Inductive miner [102] and Declare miner [37]. The first four discovery algorithms produce Petri nets, which require a suitable conformance checker. The choice has fallen on the

alignment-based *conformance-checking* technique presented in [114] that, differently from token-based algorithms [131], is able to deal with invisible transitions and duplicate activity labels. Note that the Alpha+ and Heuristics miners can yield models that are *unsound* (e.g., they contain deadlocks). In those cases, conformance checkers cannot replay traces on such models, hence the result will indicate that none of the test traces fit the discovered model. The Declare miner produces declarative process models, hence, it requires a different conformance checker. For this purpose, we chose the conformance checker presented in [42]. However, other conformance checkers for declarative models such as [19, 31] can be used instead.

Excluding the ILP miner, the other algorithms were used with the default configuration. The ILP miner was configured to generate models in which the final marking is the empty marking i.e., no remaining tokens. Any other configuration generates process models in which the ILP miner does not state what the final marking is, which would require a model inspection by a human. The human involvement would hinder the possibility of an automatic workflow.

We conducted two rounds of experiments. In both experiments, the same implemented workflow (showed in Figure 7.5) was used. However, different parameters were used in each experiment in order to tailor it towards specific purposes. The first round validates the usefulness of the proposed framework through an experiment consisting of a detailed empirical analysis of the process discovery algorithms mentioned above. The experimental setup of the first round is reported in Section 7.3.1, where the results are analyzed and discussed. In the second experiment round, the flexibility of the framework and its support for large-scale experiments are validated by extending the first round to experiments five times larger. Section 7.3.2 reports on the second (extended) round of experiments.

The implemented workflow that executes the experiments presented in this section (see Figure 7.5) is publicly available, and can be downloaded from: `https://www.dropbox.com/s/2bsyksdqwnvai47/DiscoveryBenchmark.zip?dl=0`. The reader is encouraged to download and use the workflow to reproduce the experiment results or extend the framework to other discovery algorithms or process characteristics.

7.3.1 First Experiment

As mentioned earlier, the goal of this framework is to analyze and compare the accuracy of process discovery techniques to rediscover process models based on observed executions, i.e., event logs. The population of process models that

we aim to rediscover is generated by varying a number of parameters, which identify the probability of occurrences of typical process characteristics, such as parallel branches, silent transitions and infrequent paths. Section 7.2.2 has discussed the blocks which, so far, our framework allows for and how the probabilities influence the generated process models. In the first round of experiments, the population of models is generated by varying the probability of duplicate activities and by enabling or disabling the presence of infrequent paths. In this way, we can study the impact of infrequent behavior and of different frequencies of duplicate activities on the accuracy of process discovery techniques. Section 7.3.2 will report on the extended experiment where the probability of the other process characteristics are also varied.

Therefore, the experimental design includes all the combinations of three independent variables: process discovery technique used (i.e., miner), presence or absence of infrequent paths and the probability of having *duplicate activities*. The parameter values used in this experiment are summarized in Table 7.2. In total, the 70 possible combinations are included in the experiment: 5 discovery techniques $\times$ 2 levels of infrequent behavior $\times$ 7 probabilities of duplicate activities.

Table 7.2: Summary of the possible parameter values included in the experiment: 70 ($5 \times 7 \times 2$) value combinations.

Parameter Type	**Parameter**	**Values considered**
Miner	Miner	Alpha+ [167], Declare [37], Heuristics [189], ILP [170], Inductive [102]
Model size (# act)	min	15
	mode	30
	max	60
Probability of basic c-f pattern	sequence (s)	0.46
	exclusive choice (e)	0.35
	parallelism (p)	0.19
	inclusive choice (i)	0
	loop (l)	0
Probability of advanced c-f pattern	silent transitions	0
	duplicate activities	0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3
	long-term dependency	0
	infrequent paths	true, false

As mentioned above, the other process characteristics are not taken into account in this analysis. The size of the model is defined by a fixed triangular distribution from 15 to 60 activities, with a mode of 30 activities per model. The probability of non-exclusive choice (OR) and of loops are set to zero and, hence, these two constructs do not occur. The probability of sequence, exclusive choice and parallelism is set and kept fixed to values 46%, 35% and 19%, respectively. These values have been determined after analysing their frequencies in the large collections of models reported in [98]. In this work, Kunze et al. have observed that 95% of the models consist of activities connected in sequences, 70% of the models consist of activities, sequences and XOR connectors and 38% consist of sequences, activities and AND connectors (see Fig. 4b of the paper). Assuming independence of occurrence probability of sequences, AND and XOR, it follows that:

$$P(sequence) = 0.95$$
$$P(sequence \wedge XOR) = P(sequence) \times P(XOR) = 0.70 \Rightarrow P(XOR) = 0.74$$
$$P(sequence \wedge AND) = P(sequence) \times P(AND) = 0.38 \Rightarrow P(AND) = 0.4$$

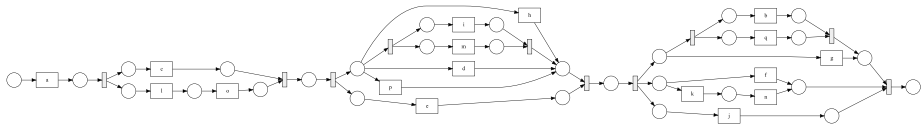
When these values are normalized to 1, the final probabilities of the constructs are obtained.

For each discovery technique a random sample of 62 process models is drawn. The sample size of 62 models allows us to study the effect of process discovery techniques, infrequent paths and different probabilities of duplicate-activity occurrences (and their interactions) using a fixed effects ANOVA analysis [48] with significance level $\alpha = 0.05$ and power $1 - \beta = 0.98$. This power indicates the probability to detect a significant effect when two mining algorithms actually differ by a relatively small difference. In total 4340 process models (70 settings $\times$ 62 models per setting) were generated.

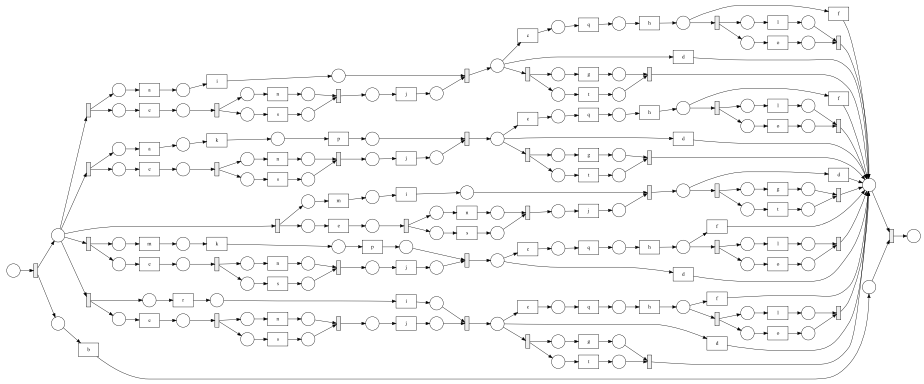
Figures 7.6a and 7.6b show two examples of the generated models in a Petri net notation: the first shows a relatively simple model with no duplicate activities, while the second shows a larger model with plenty of duplicate activities.

For each of the obtained process models, an event log containing between 200 and 1000 traces is generated (See Section 7.2.2). For each generated log, we can calculate the completeness i.e., the ratio of unique traces in the log to all possible unique traces according to the model using the technique described in [76]. Figure 7.7 shows a histogram of the completeness of the event logs generated in this experiment.

The effect of process-discovery techniques, infrequent paths and different probabilities of duplicate-activity occurrences can be analyzed using one-way ANOVA analysis if the assumptions of homogeneity of variances and normality



(a) Simpler model without duplicate activities.



(b) More complex model with duplicate activities.

Figure 7.6: Samples of the models generated in this experiment.

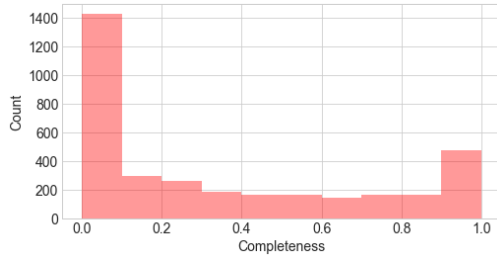


Figure 7.7: Distribution of completeness of logs wrt. their respective process models. Completeness is measured as the fraction of traces allowed by the model that are present in the event log.

of the dependent variable hold [48]. However, both assumptions were violated for every dependent variable i.e., F_1, recall and precision. Therefore, the non-parametric *Kruskall-Wallis* test (KW) [136] was applied instead.

KW is used for testing whether k independent samples are from different

populations. It starts by ranking all the data from the different samples together: assign the highest score a rank 1 and the lowest a rank N, where N is the total number of observations in the k samples. Then, the average ranking for each sample is computed, e.g. the mean of sample j is denoted as $\bar{R}_j$. With n the number of observations in each sample, the test statistic KW, which follows a χ^2 distribution with $k-1$ degrees of freedom, can be calculated as: KW $= [\frac{12}{(N(N+1))} \sum_{j=1}^{k} n\bar{R}_j^2] - 3(N+1)$. If the calculated KW is significant, then it indicates that at least one of the samples is different from at least one of the others. Subsequently, the multiple comparison post hoc test is applied to determine which samples are different. More specifically, for all pairs of samples R_i and R_j it is tested whether they differ significantly from each other using the inequality: $|R_i - R_j| \geq z_{\alpha/k(k-1)} \sqrt{\frac{N(N+1)}{12}(\frac{2}{n})}$. The $z_{\alpha/k(k-1)}$ value can be obtained from a normal distribution table given a significance level α. The formula adjusts this α with a Bonferroni correction to compensate for multiple comparisons. If the absolute value of the difference in average ranks is greater than or equal to the critical value i.e., the right side of the equation, then the difference is significant.

Finally, the Jonckheere test [136] can be used to test for a significant trend between the k samples. First, arrange the samples according to the hypothesized trend, e.g. in case of a positive trend from smallest hypothesized mean to highest hypothesized mean. Then count the number of times an observation in sample i precedes an observation in sample j, denoted as U_{ij} $\forall i < j$. The Jonckheere test statistic J is the total number of these counts: J $= \sum_{i<j}^{k} U_{ij}$. When J is greater than the critical value for a given significance level α, then the trend between the k samples is significant.

The Effect of Process Discovery Technique

The goal is to learn the effect of a process discovery technique on each of the dependent variables: recall, precision and F_1 score. The other variables (i.e., infrequent paths level and probability of duplicate activities) are part of the error term.

We apply the KW method, to test whether the average rank differs between the five process discovery techniques (i.e., samples). In this case we ranked all the 4340 averages over a 10-fold cross validation for recall, precision and F_1 score values ignoring sample membership (i.e. discovery technique). Note that the samples are ranked only after all the recall, precision and F_1 score values have been calculated. The highest value for recall, precision and F_1 score gets

rank 1 (lowest rank), while the lowest absolute value gets rank 4340 (highest rank).

Then we computed the average ranking per miner i.e., the average position of a discovered model by that miner for that quality metric. The samples are ranked all together on a scale from 1 to 4340. Then, for each miner, we select the samples that were used with such miner, and calculate their average rank for each quality metric. A higher average ranking means worse performance. The ranking summary is shown in Table 7.3.

Table 7.3: Average ranks per miner (n = 4340). Each cell indicates the average ranking for a specific performance dimension (row header) and for a specific miner (column header). One can compare miners by comparing the average ranks within one row.

	Alpha+	**Declare**	**Heuristics**	**ILP**	**Inductive**
Recall	2912.37	2424.87	3299.76	558.54	1656.94
Precision	2649.08	2605.64	3261.80	1060.34	1275.62
F_1 **score**	2851.28	2535.53	3293.33	727.16	1445.18

Based on the average rankings in Table 7.3, the order suggested between process discovery techniques is: ILP $\prec$ Inductive $\prec$ Declare $\prec$ Alpha+ $\prec$ Heuristics for recall, precision and F_1 scores. It indicates that the ILP miner creates the best models in terms of recall, precision and F_1 scores. The Inductive miner outperforms the Declare and Alpha+ miners, which in turn outperform the Heuristics miner. The results of the KW test confirm that the differences in average rankings between the five miners are statistically significant (significance level $\alpha = 0.05$). Moreover, the multiple comparison post-hoc test (cf. supra) also confirms the statistical significance of the differences between algorithms, as shown in Table 7.4.

The Effect of Infrequent Paths

This analysis tests whether the presence/absence of infrequent paths[3] has an impact on the average ranking of the five process discovery techniques for recall, precision and F_1 scores. The effect of duplicate activities is a part of the

[3]Infrequent paths are denoted with an imbalance in execution probabilities of the output-branches of each exclusive choice construct in the model which results in an event log containing infrequent behavior.

Table 7.4: Results of the statistical tests to study the effect of discovery algorithms on F_1 scores.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 2691.8$	degrees of freedom $= 4$	p-value $< 2.2e^{-16}$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
Alpha+ - Declare	315.75	168.83	True
Alpha+ - Heuristics	442.05	168.83	True
Alpha+ - ILP	2124.12	168.83	True
Alpha+ - Inductive	1406.10	168.83	True
Declare - Heuristics	757.80	168.83	True
Declare - ILP	1808.37	168.83	True
Declare - Inductive	1090.35	168.83	True
Heuristic - ILP	2566.17	168.83	True
Heuristic - Inductive	1848.15	168.83	True
ILP - Inductive	718.02	168.83	True

error term. Firstly, the sample of size 4340 is split into two subsets: 2170 samples with infrequent behavior and 2170 samples without infrequent behavior. For each subset, the samples were ranked from 1 to 2170 according to their precision, recall and F_1 score values. This division is called *blocking* (see Section 7.2) which is done to isolate the variation in recall, precision and F_1 scores attributable to the absence/presence of infrequent paths. Secondly, the KW test is applied to each subset. Table 7.5 contains the average rankings per process discovery technique grouped by metric and experiments with and without infrequent behavior. These rankings again suggest the same order between process discovery techniques in all cases: ILP $\prec$ Inductive $\prec$ Declare $\prec$ Alpha+ $\prec$ Heuristics.

In the absence of infrequent behavior, multiple comparison post-hoc tests show that all the miners have statistically-significant differences with all the other miners. However, this is not the case in the presence of infrequent behavior, as the multiple comparison post-hoc test shows that the Alpha+ and Declare miners do not have a statistically-significant difference in terms of F_1 scores. Table 7.6 shows a summary of the statistical post-hoc test results for the F_1 scores in the presence of infrequent behavior.

Note that only the difference between the Alpha+ and Declare miners in the

Table 7.5: Average ranks per miner in terms of recall, precision and F_1 Scores.

(a) Without infrequent behavior (n=2170)

	Alpha+	**Declare**	**Heuristics**	**ILP**	**Inductive**
Recall	1490.35	1144.75	1670.96	296.58	824.83
Precision	1310.13	1322.56	1639.10	507.50	648.18
F_1 **score**	1461.35	1232.10	1669.64	337.78	726.61

(b) With infrequent behavior (n=2170)

	Alpha+	**Declare**	**Heuristics**	**ILP**	**Inductive**
Recall	1424.36	1267.79	1631.71	258.24	845.37
Precision	1333.31	1291.56	1621.94	551.55	629.12
F_1 **score**	1394.88	1277.99	1628.01	383.95	742.65

Table 7.6: Results of the statistical tests to study the effect of the miner on F_1 scores in the presence of infrequent behavior.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 1261.8$	degrees of freedom $= 4$	p-value $< 2.2e^{-16}$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
Alpha+ - Declare	116.88	119.39	False
Alpha+ - Heuristics	233.12	119.39	True
Alpha+ - ILP	1010.92	119.39	True
Alpha+ - Inductive	652.22	119.39	True
Declare - Heuristics	350.01	119.39	True
Declare - ILP	894.04	119.39	True
Declare - Inductive	535.33	119.39	True
Heuristic - ILP	1244.05	119.39	True
Heuristic - Inductive	885.35	119.39	True
ILP - Inductive	358.70	119.39	True

presence infrequent behavior is not statistically significant for F_1 scores. Therefore, one cannot accept the assumption that infrequent paths do not influence process discovery techniques.

The Effect of Duplicate Activities

The analysis investigates how the accuracy of each process discovery technique (in terms of precision, recall and F_1 score) is influenced by the probability of duplicate activities (i.e., the average percentage of duplicated visible activity labels in the process models). The effect of infrequent behavior is a part of the error term. Figure 7.8 illustrates the average F_1 scores for all the process discovery techniques over different probabilities of duplicate activities.

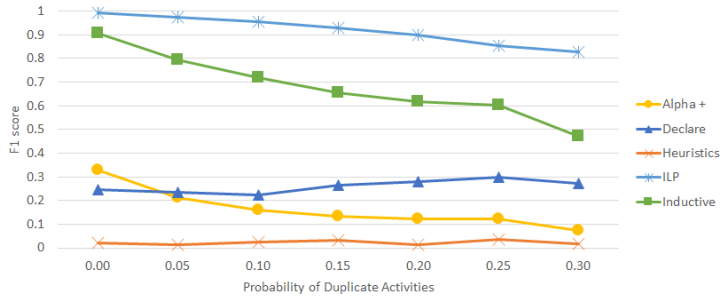


Figure 7.8: F_1 scores for process discovery techniques for different probabilities of duplicate activities

This graph indicates a general negative trend i.e., the probability of duplicate activities has a negative effect on F_1 scores. To determine whether such a trend is statistically significant, an in-depth analysis is performed. First, the 4340 samples are divided into five subsets of 868 samples each, grouped by the process discovery technique. The samples of each subset were ranked from 1 to 868 according to their precision, recall and F_1 score values. As such, the variation in accuracy associated with the discovery technique is isolated. Then, similar to the analysis above, the KW test is applied to compare the average rankings of the discovered models. Table 7.7 contains one subtable for each process discovery technique with the average ranks for all three metrics by the probability of duplicate activities.

For the Alpha+ miner, the data (shown in Table 7.7a) seems to suggest that as the probability of duplicate activities increases, the models generated by Alpha+ miner deteriorate in terms of recall, precision and F_1 score. To test this impression statistically, we will rely on the KW and Jonckheere tests. The results of these statistical tests and the multiple pair-wise comparison between different probabilities of duplicate activities for the Alpha+ miner is presented

Table 7.7: Average ranks of process discovery techniques per probability of duplicate activities in terms of recall, precision and F_1 scores.

(a) Alpha+ miner (n=868)

Prob. Duplicate Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	339.47	417.21	428.08	451.02	435.54	479.52	490.66
Precision	346.37	421.59	423.21	449.60	431.10	478.01	491.62
F_1 **score**	339.46	417.56	427.86	450.78	435.69	479.36	490.78

(b) Declare miner (n=868)

Prob. Duplicate Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	456.86	451.45	462.90	428.41	427.99	393.00	420.86
Precision	450.21	442.10	457.47	432.26	440.84	382.23	436.36
F_1 **score**	451.70	445.89	460.29	431.36	431.54	393.97	426.71

(c) Heuristics miner (n=868)

Prob. Duplicate Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	432.08	429.27	435.26	428.28	442.85	437.97	435.79
Precision	431.96	429.40	434.90	428.14	442.76	438.51	435.83
F_1 **score**	432.04	429.34	435.19	428.27	442.85	438.06	435.75

(d) ILP miner (n=868)

Prob. Duplicate Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	466.09	438.65	457.05	462.85	416.74	418.97	381.15
Precision	172.53	295.21	369.48	431.76	521.84	604.08	646.60
F_1 **score**	185.64	288.92	370.51	430.34	519.42	602.04	644.64

(e) Inductive miner (n=868)

Prob. Duplicate Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	239.90	339.92	415.75	462.62	492.83	515.94	574.54
Precision	241.90	331.42	405.01	478.28	496.85	490.82	597.22
F_1 **score**	219.71	339.45	410.69	474.73	500.88	505.77	590.27

in Table 7.8. Both tests show that there is statistically significant negative trend in the relative quality of the generated models as the probability of duplicate activities increases. A pairwise comparison of each probability of duplicate ac-

Table 7.8: Results of the statistical tests to study the effect of duplicate activities on F_1 scores for the Alpha+ miner.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 44.29$	degrees of freedom $= 6$	p-value $< 6.485e^{-8}$	
Jonckheere-Terpstra test			
JT $= 140560$		p-value $= 0.0002$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	78.10	96.73	False
0.0-0.1	88.40	96.73	False
0.0-0.15	111.32	96.73	True
0.0-0.2	96.23	96.73	False
0.0-0.25	139.90	96.73	True
0.0-0.3	151.32	96.73	True
0.05-0.1	10.30	96.73	False
0.05-0.15	33.22	96.73	False
0.05-0.2	18.13	96.73	False
0.05-0.25	61.80	96.73	False
0.05-0.3	73.21	96.73	False
0.1-0.15	22.92	96.73	False
0.1-0.2	7.83	96.73	False
0.1-0.25	51.50	96.73	False
0.1-0.3	62.92	96.73	False
0.15-0.2	15.09	96.73	False
0.15-0.25	28.58	96.73	False
0.15-0.3	40.0	96.73	False
0.2-0.25	43.67	96.73	False
0.2-0.3	55.09	96.73	False
0.25-0.3	11.42	96.73	False

tivities does not provide a clear picture how this trend looks like for recall, with many comparisons statistically insignificant. For precision and F_1 on the other hand, the quality of the models decreases significantly whenever the probability of duplicate activities increases from 0% to more than or equal to 15%.

For the Declare miner, increasing probabilities of duplicate activities seems

to have a positive impact on F_1 scores (see Table 7.7b). However, the KW and Jonckheere tests, presented in Table 7.9, confirm that there is no evidence of a trend in recall, precision and F_1 score as the probability of duplicate activities increases.

Table 7.9: Results of the statistical tests to study the effect of duplicate activities on F_1 scores for the Declare miner.

Kruskall-Wallis rank sum test		
KW $\chi^2 = 6.099$	degrees of freedom $= 6$	p-value $= 0.4122$
Jonckheere-Terpstra test		
JT $= 168806$	p-value $= 0.066$	

The models discovered using the Heuristics miner seem insensitive to the probability of duplicate activities (see Table 7.7c). The KW and Jonckheere tests, presented in Table 7.10, confirm that there is indeed statistically insufficient evidence of a trend in recall, precision and F_1 score as the probability of duplicate activities increases.

The results for the ILP miner in Table 7.7d suggest a positive trend in the probability of duplicate activities in terms of recall! This will be discussed later in this section. However, in terms of precision, the ILP miner shows high sensitivity to the probability of duplicate activities. The KW and Jonckheere tests confirm both statements, as shown in Table 7.11. The pairwise comparisons of duplicate activities presented in this table reveals the significant negative trend in terms of F_1 scores of the generated models as the probability of duplicate activities increases.

The findings for the Inductive miner indicate that as the probability of duplicate activities increases, the model quality in terms of recall, precision and F_1 score deteriorates, as shown in Table 7.7e. This effect, though, seems to level off as we reach higher probabilities of duplicate activities. The KW and Jonckheere tests show that there is indeed a significant negative trend in the relative quality of the generated models as the probability of duplicate activities increases, as shown in Table 7.12. However, at a probability of around 15% of duplicate activities, this effect seems to have reached a plateau and stays stable.

7.3.2 Second (Extended) Experiment

The above experiments have validated the usefulness of the proposed evaluation framework to support the benchmark and sensitivity analysis evaluation

Table 7.10: Results of the statistical tests to study the effect of duplicate activities on F_1 scores for the Heuristics miner.

Kruskall-Wallis rank sum test		
KW $\chi^2 = 1.4786$	degrees of freedom $= 6$	p-value $= 0.9609$
Jonckheere-Terpstra test		
JT $= 160165$	p-value $= 0.255$	

Table 7.11: Results of the statistical tests to study the effect of duplicate activities on F_1 scores for the ILP miner.

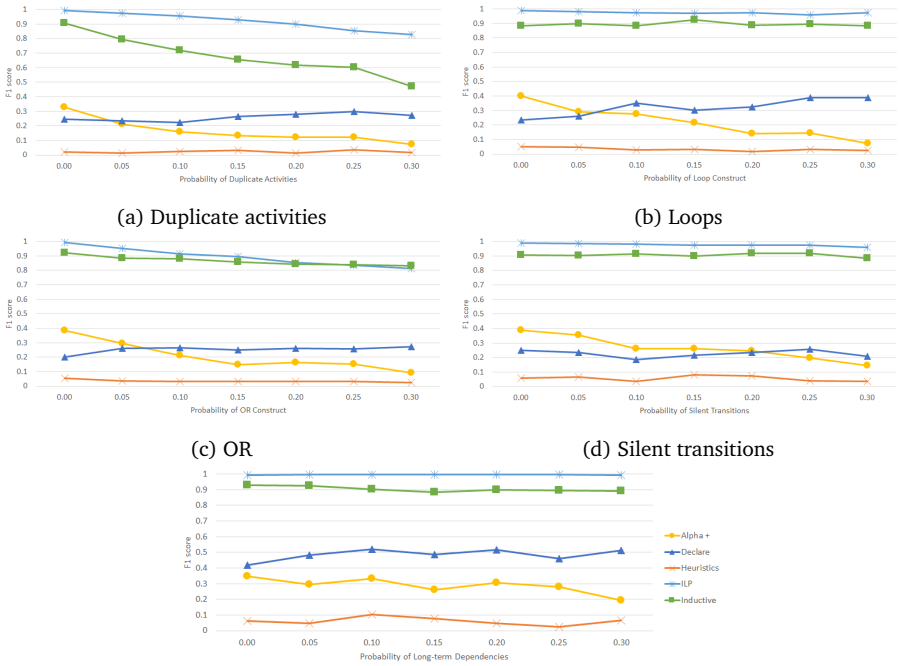
Kruskall-Wallis rank sum test			
KW $\chi^2 = 331.81$	degrees of freedom $= 6$	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT $= 81029$		p-value $= 0.0002$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	103.28	96.73	True
0.0-0.1	184.87	96.73	True
0.0-0.15	244.71	96.73	True
0.0-0.2	333.78	96.73	True
0.0-0.25	416.40	96.73	True
0.0-0.3	459.0	96.73	True
0.05-0.1	81.59	96.73	False
0.05-0.15	141.42	96.73	True
0.05-0.2	230.50	96.73	True
0.05-0.25	313.12	96.73	True
0.05-0.3	355.72	96.73	True
0.1-0.15	59.83	96.73	False
0.1-0.2	148.91	96.73	True
0.1-0.25	231.53	96.73	True
0.1-0.3	274.13	96.73	True
0.15-0.2	89.07	96.73	False
0.15-0.25	171.70	96.73	True
0.15-0.3	214.29	96.73	True
0.2-0.25	82.63	96.73	False
0.2-0.3	125.22	96.73	True
0.25-0.3	42.60	96.73	False

Table 7.12: Results of the statistical tests to study the effect of duplicate activities on F_1 scores for the Inductive miner.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 180$	degrees of freedom $= 6$	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT $= 105512$		p-value $= 0.0002$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	119.75	96.73	True
0.0-0.1	190.98	96.73	True
0.0-0.15	255.02	96.73	True
0.0-0.2	281.17	96.73	True
0.0-0.25	286.07	96.73	True
0.0-0.3	370.57	96.73	True
0.05-0.1	71.24	96.73	False
0.05-0.15	135.27	96.73	True
0.05-0.2	161.43	96.73	True
0.05-0.25	166.32	96.73	True
0.05-0.3	250.82	96.73	True
0.1-0.15	64.04	96.73	False
0.1-0.2	90.19	96.73	False
0.1-0.25	95.08	96.73	False
0.1-0.3	179.58	96.73	True
0.15-0.2	26.15	96.73	False
0.15-0.25	31.05	96.73	False
0.15-0.3	115.55	96.73	True
0.2-0.25	4.90	96.73	False
0.2-0.3	89.40	96.73	False
0.25-0.3	84.50	96.73	False

objectives. The proposed framework is also flexible as it allows users to easily setup extended experiments. Here, we have extended the above experiment with other control-flow characteristics. The probability of the basic characteristics, sequence, parallel and exclusive choice, is set the same as in the previous experiments. In this experiment, for each process characteristic, we have varied the probability of its occurrence while setting the probability of the others to zero. Instead of 4340 observations as in the first experiment, the extended experiment results in 21700 observations. Figure 7.9 illustrates how the differ-

ent algorithms perform in terms of F_1 score with varying probabilities of the constructs.



(a) Duplicate activities

(b) Loops

(c) OR

(d) Silent transitions

(e) Long-term dependencies

Figure 7.9: F_1 scores for process discovery techniques for different probabilities of process control-flow characteristics.

It is clear that the ILP and Inductive miner perform significantly better than the Alpha+, Declare and Heuristics miner. In fact, this is not surprising because the Alpha+ and Heuristics miners are not guaranteed to produce sound models, which allow executions to be carried out till completion. Models discovered with Alpha+ and Heuristics miner can contain deadlocks, livelocks, and other anomalies [156]. When a model is indeed not sound, it cannot replay traces until the end and, hence, the confusion matrix may contain few true positives (often none), causing precision, recall and F_1 scores to be very low (often zero). This is not trivial because, although the theory already postulated it, it was not clear how much the lack of soundness guarantee was practically affecting the

results. Ultimately this means that the Alpha+ miner, Declare miner and Heuristics miner can be useful to gain an initial insight into the general structure of the process but should not be used for more precise analysis such as conformance checking.

Looking at Fig. 7.9, the ILP miner tends to perform better than Inductive miner in terms of F_1 score. This is observed for all constructs and all occurrence probabilities considered in this experiment. In particular, for such constructs as silent transitions and long-term dependencies, the F_1 score is steadily around 1, which indicates almost perfect precision and recall. This result is far from being trivial: as discussed in [170], the ILP miner focuses on producing models that can replay every trace of the event log, without trying to maximize precision. Furthermore, because ILP miner only aims at replaying the traces in the event log used for discovery, one would expect that a different event log, e.g., used for testing, would not let the discovered models score high in recall, either.

These findings are supported by visually comparing the models that the ILP miner generates and those from the Inductive miner, such as the models presented in Figures 7.10 and 7.11 respectively discovered through the Inductive and ILP miner.

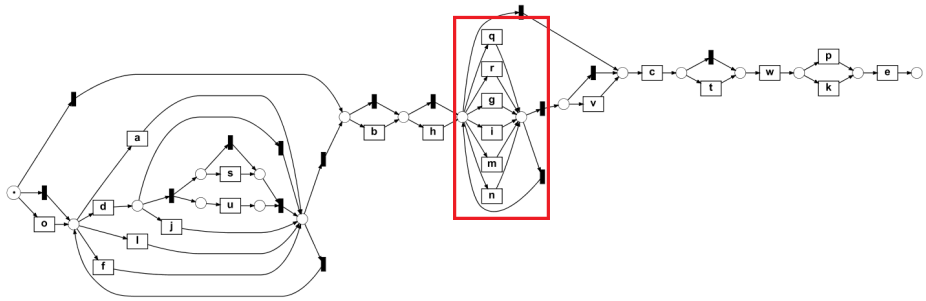


Figure 7.10: Model discovered by the Inductive miner.

The red boxes in these figures illustrate the unprecise parts of the model. For the Inductive miner model, the transitions in the box can be executed in any order and, because of the loop, an arbitrary number of time. Of course, in the reality, these transitions should occur in a more precise order; but the miner is unable to "see it". Conversely, for the model discovered through the ILP miner, the only "source of imprecision" is related to the "floating transition" a but it is just one out of 26 transitions. As discussed in Section 7.2.2, to punish for imprecise behavior, our framework injects noise into fitting traces. In case

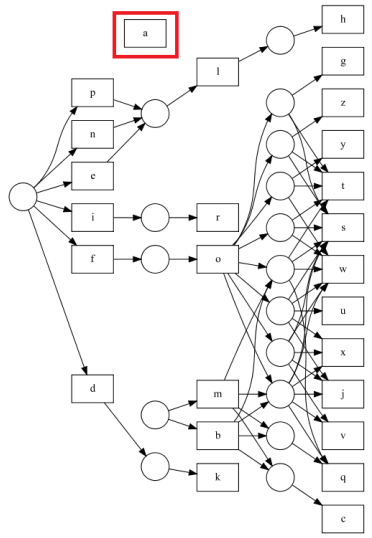


Figure 7.11: Model discovered by the ILP miner.

of the models by the ILP miner, the probability that the noise would involve the only "floating transition" a is low. However, the probability that noise affects activities present in precise regions of the model is high. Such deviations in very precise regions are easily detected, resulting in high F_1 scores for the ILP miner. The same reasoning is shared among most of models.

Another interesting result for both the Inductive miner and the ILP miner is that the values of F_1 score seem not to be really affected by the amount of occurrences of the process constructs, except for duplicate activities and, limitedly, from the OR construct. The OR is known to be a hard construct and neither of the two miners provides specific support for it (for Inductive miner, at least for the version being evaluated). For duplicate activities, this can be explained by the fact that both ILP and Inductive miner do not natively support mining models where different transitions share the same activity label. This means that duplicate activities are causing the creation of loops: a transition without input places is a loop (see above), which would underfit the behavior observed in the event log, thereby yielding lower precision.

7.4 Conclusions

Existing empirical evaluation frameworks for process discovery techniques have several important drawbacks. This chapter presented a new evaluation framework to overcome the existing limitations. The framework allows researchers to benchmark discovery algorithms as well as to perform a sensitivity analysis to evaluate whether certain model or log characteristics have a significant effect on an algorithm's performance. It is independent from the discovered model's modeling notation by adopting a classification approach that uses the knowledge of the original (reference) model to be rediscovered. Additionally, the framework allows to generalize the evaluation results to a user specified model population taking into account the size of the event logs, the applied noise operators and parameter settings of the employed discovery algorithms. Finally, the design of the framework as a process mining workflow enables automating, sharing and extending evaluation experiments.

The framework has been validated by conducting an extensive experiment involving five process discovery algorithms that produce models in different notations, five control-flow characteristics and two levels of infrequent behavior. The experiment has shown the usefulness and flexibility of the framework. Additionally, the analysis of the experiment results has led to several non-trivial insights on discovery algorithms in the context of the populations of processes included in the experiments. Firstly, the evaluation results have illustrated that the inability of the Alpha+ and Heuristics miner to always discover sound models strongly affects the quality results. Secondly, the ILP miner handled the incompleteness of the event logs well as it discovered models with high recall and precision. Thirdly, the amount of occurrences of loops, silent transitions and long-term dependencies have only slightly affected the recall and precision scores of both the Inductive and ILP miner. Finally, the results indicate that the ILP and Inductive miner greatly outperform the Alpha+, Declare and Heuristics miner in all the considered scenarios.

As limitations of this framework, we have to mention that this experiment was conducted was conducted over populations of block-structured processes, and that specific types of noise were used. Therefore, future work aims at extending the populations that can be generated for further experimenting with even more noise types.

Chapter 8

A Framework for Benchmarking Concept Drift Detection Techniques

Processes evolve and change over time. This is only natural as they have to adapt to a dynamic context that includes new laws and regulations, changing customer preferences and demands and new competitors.

In the context of process mining, a *concept drift* is described as a change in a process over time. It is important to detect concept drifts in event data in order to obtain simpler, more precise and more realistic models out of event data, so that they can be used to obtain new insights about the process (see Challenge "*Dealing with concept drift*" in [152]).

As discussed in Chapter 1, concept drifts can also be considered as a form of *process variability* where different *variants* of the process are distributed over time. Any change in the process can be related to different variants of the process e.g., one describing the process *before* and one describing the process *after* the change. As an example, consider that processes evolve over time: activities that were once necessary can become obsolete and are removed from the process. Also, new activities can be incorporated into the process as a result of new regulations. As a consequence, the difficulty of analyzing a process can be related to the amount and impact of changes in it: the more changes a process has had, the more difficult it is to analyze. It is unlikely that a model

that represents the behaviour of several different process variants combined will have a better precision than a collection of models where each represents a different process variant.

Concept drift detection is not new: These techniques have been developed since the 80's. Many existing data mining techniques can be used to detect concept drift e.g., support vector machines, and sliding windows. See [56] for an extensive survey on concept drift detection techniques in the context of data mining. There are many applications of concept drift techniques to a range of problems. The work presented in [195] discusses an exhaustive taxonomy of concept drift applications to various types of problems in data mining.

Concept drift detection techniques have also been proposed in the context of process mining. However, these approaches do not perform an exhaustive comparison with respect to other techniques. Some of these techniques do a comparison over a small and ad-hoc collection of event logs obtained from manually-designed process models, where drift is manually inserted. There is the need for a framework that allows the objective and large-scale comparison of concept drift techniques in process mining.

Moreover, there is no standard way to evaluate concept drift approaches in process mining. For example, sliding-window-based approaches are evaluated by the size of the window that is needed to detect a drift, while other approaches (e.g., decision-tree-based) cannot use such metrics. Therefore, it is also important to unify and standardize the way that concept drift techniques are evaluated in a process mining setting, such that the evaluation is compatible with all present and future concept drift detection techniques.

This chapter proposes a framework for benchmarking concept drift detection techniques in the form of a process mining workflow (see Chapter 4) that allows a standardized evaluation and an exhaustive comparison of concept drift detection techniques in the context of process mining.

The remainder of this chapter is structured as follows. Section 8.1 elaborates on the concept drift and discusses related work. Section 8.2 proposes a novel framework for benchmarking concept drift techniques in process mining in the form of a process mining workflow and describes the building blocks used in it. Section 8.3 discusses the implementation of the framework and provides a description of the experiments. Then, the obtained results are discussed. Finally, Section 8.4 concludes the chapter.

8.1 Related Work

The term *concept drift* [134] is used in the field of data mining to generically describe *changes in data over time*. The underlying assumption is that the system that generates such data is subject to changes, and when the system changes, the observed behavior of the system (i.e., the data) will reflect such changes. Since the systems and their explicit changes are often not available and well-defined, concept drift techniques aim to discover such changes from the observed data itself.

The work presented in [20] proposes three challenges in concept drift detection: *change point detection* (i.e., detect that a process change has taken place), *change localization and characterization* (i.e., to characterize the nature of change, and identify the region(s) of change (localization) in a process), and *change process discovery* (i.e., the discovery of the change process describing the second-order dynamics). It also proposes two classes of concept drift detection: *online* (i.e., changes need to be discovered in near real time) and *offline* (i.e., the presence of changes or the occurrence of drifts do not need to be uncovered in a real time). Moreover, Bose [20] defines four types of concept drift: *sudden drift* (i.e., a substitution of a process by another variant of the process, where both processes do not coexist in time), *gradual drift* (i.e., a substitution of a process by another variant of the process, where both processes do coexist for some time, and the first process is gradually discontinued), *recurring drift* (i.e., a set of process variants reappear after some time) and *incremental drift* (i.e., a drift is composed by several small incremental drifts).

In this chapter, we consider recurring and incremental drifts simply as a composition of sudden and/or gradual drifts. Hence, only sudden and gradual drifts will be considered in this chapter. Figure 8.1 illustrates these two types of concept drift.

In the context of data mining, the work presented in [4] proposes an evaluation criteria to compare concept drift detection techniques based on the well-known concepts of precision, recall and F_1 score, and applies it to several techniques using a small collection of datasets. To the best of our knowledge, there are no available frameworks for benchmarking concept drift detection techniques in a process mining context.

In the context of process mining, several concept drift techniques have been developed in recent years [20, 33, 109, 111, 112, 116]. Even the process variant detection technique presented in Chapter 6 can be used to detect concept drift (i.e., by using the "next activity" attribute as the dependent variable and the "timestamp" attribute as the independent variable). All these techniques

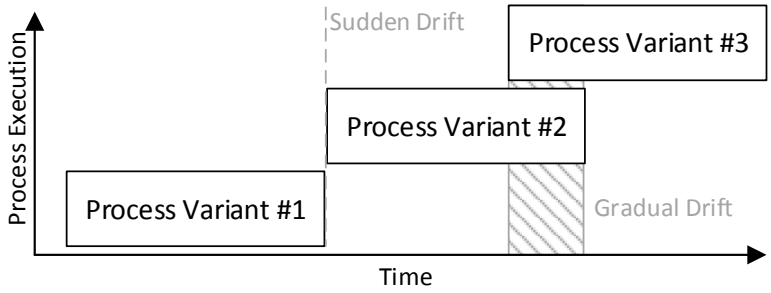


Figure 8.1: Types of concept drift in processes

work nicely and are evaluated using a few sample event logs simulated from manually-designed models. However, since most of these techniques only focus on *change point detection*, we will only focus on change point detection techniques in an *offline* setting in order to provide a "fair" benchmarking framework. It is important to note that online techniques can also work in an offline setting, but not the other way around.

8.2 Concept Drift Evaluation Framework

The framework presented in this chapter aims to evaluate the ability of process-oriented concept drift detection algorithms to detect control-flow changes in processes. Similarly to the process discovery benchmarking framework presented in Chapter 7, this framework is designed based on the principles of scientific workflows and experimental design. The former captures the complete evaluation experiment in a workflow that can be automated, reused, refined and shared with other researchers [8]. Such advantages were discussed in detail in Section 7.2. The latter allows for precise answers that a researcher seeks to answer with the evaluation experiment [93]. As described in Section 7.2, the three cornerstones of good experimental design are: randomization, replication and blocking [55]. The three cornerstones together are fundamental to make the experiments scientifically sound (e.g., avoid bias or wrong conclusions). Therefore, the framework proposed in this chapter incorporates each of

the cornerstones.

To integrate the steps needed for empirically evaluating process-oriented concept drift detection algorithms, the framework is built as a *process mining workflow* (see Chapter 4).

The remainder of the section will discuss the design of the framework (Section 8.2.1) and its building blocks (Section 8.2.2) in more detail.

8.2.1 The Design of the Framework

The framework focuses on evaluating control-flow concept drift detection algorithms. Therefore, other process related perspectives (e.g., data and resources) are not presented here, but can of course be supported in a similar way. Moreover, the framework aims at evaluation instead of predicting the best performing algorithm given an event log. The framework enables two main objectives: either benchmarking different concept drift detection algorithms, or performing sensitivity analysis e.g., what effect does a control-flow characteristic or event log characteristic have on algorithm performance.

Figure 8.2 illustrates the framework instantiating the "large-scale experiment" use case (see Section 4.4.3). The framework uses the *process mining*

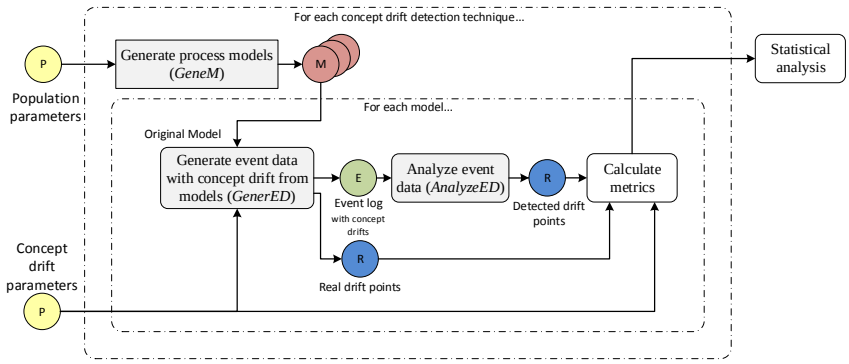


Figure 8.2: Framework for concept drift detection algorithm evaluation, presented as an analysis scenario. Grey boxes represent process mining building blocks. White boxes represent non-process-mining operators.

building blocks defined in Section 4.2, represented as grey boxes. It also con-

tains non-process-specific blocks, represented as white boxes. Such blocks represent generic data-processing functionalities (e.g., calculate metrics) that complement the process mining building blocks in order to obtain the desired analysis results.

The framework enforces the consecutive execution of data generation, concept drift detection, quality measurement and statistical analysis.

Similarly to Chapter 7, the first step i.e., the data generation, is triggered by the objective of the experiment. As a result, the objective determines the desired control-flow behavior to be included in the event logs. The specification of control-flow behavior defines a population of process models. This population definition is the start of the data generation phase.

For each concept drift detection algorithm to be tested, multiple instances of the "generate process models" block run in parallel. The generation results in multiple *random* samples of process models from the same population.

Each model ("original model") is then used by the block "generate event data with concept drift from models" to create one event log (i.e., a *random* sample of traces from all possible traces allowed by the model) that contains *concept drift* of the behavior in the process, according to some parameter definition. The results of this block are an event log with concept drift, and a set of drift points i.e., points in time when a drift was forced. The samples of process models, event logs and drift points constitute the *ground truth*.

Next, in the block "analyze event data" an algorithm to detect concept drift is applied in each event log. The result of this block is a set of "discovered" drift points in time for which the technique detected that there was a concept drift.

Subsequently, the framework measures the quality of the concept drift detection algorithm by comparing, for each event log, the drift points "discovered" by the algorithm with the original drift points used to generate the event log. These results are then combined in a confusion matrix in the block "calculate metrics". This is discussed in the next section. Based on that matrix, one can compute the well-known *recall* and *precision* metrics to evaluate the quality of the concept drift detection algorithms.

Finally, the block "statistical analysis" tests the hypotheses formulated in the context of the objectives.

Similarly to Chapter 7, this framework's design has the property that no two concept drift detection techniques are applied on the same event log in order to reduce the selection bias. Furthermore, for each generated model - randomly drawn from a predefined population of models - we generate only a single event log that includes concept drift. Consequently, all discovered concept drifts and any corresponding quality metrics are independent observations, which is an

important assumption made by many standard statistical techniques.

Finally, the framework's design based on the experimental design principles enables the users to obtain algorithm's performance measures that are *independent* from specific process models and event logs. More specifically, this starts from the generation of (preferably large[1]) random samples of process models and logs from a population, which are then used to estimate the performance of an algorithm with regard to that population. This contrasts with evaluations based on a small non-random sample of (manually created) process models and event logs as it could influence the performance estimate to only reflect these particular models and logs.

8.2.2 Building Blocks

The previous section described the design of the framework, and briefly described the blocks that conform it. This section elaborates on each of the blocks presented above.

Generate process models

This building block was already introduced in Chapter 7. Hence, the reader is referred to Section 7.2.2 for more details about this building block and its parameters. As a summary, this building block generates a random sample of process models from a population of models. The input of this block is a set of parameters describing the population characteristics. The user can specify such population characteristics by assigning probabilities to control-flow characteristics of the model (e.g., parallelism) and setting the size of the models in terms of visible activities, through the use of the same parameters defined in Table 7.1, presented in Section 7.2.2.

In this framework (similarly to Chapter 7) process models are also generated as *process trees* [158], which support for all the constructs/patterns mentioned in Section 7.2.2 e.g., parallelism, loops, inclusive and exclusive choice. To feature the artificial, random generations of process trees, the framework applies the technique and implementation reported in [88], as mentioned in Section 7.2.2. Concretely, the user defines a population of process models by setting values for the parameters of this block, as described in Table 7.1.

[1]One can define the required sample size based on the desired power of the statistical analysis in advance, see [38].

Generate event data with concept drift from models

For each generated model, this block creates an event log with concept drift based on a set of parameters that describe the characteristics of the concept drifts to be included in the resulting event log. The input of this block is a process model (i.e., the "original" model) and a set of parameters. Concretely, the user can define the following parameters for generating an event log with concept drift:

Table 8.1: Parameters used to create an event log with concept drift.

Parameter Type	Parameter	Value set
Model changes	Number of drifts	$\mathbb{N}$
	Type of change	{add, remove, swap fragments}
Drift properties	Type of drift	{sudden, gradual}
	Duration of stable period	$\mathbb{R}^+$
	Duration of drift period (*gradual drift*)	$\mathbb{R}^+$
	Transition function (*gradual drift*)	{linear, exponential}
Event log density	Time between cases	$\mathbb{R}^+$
	Time between events	$\mathbb{R}^+$
Accuracy	Epsilon	$\mathbb{R}^+$

The functionality of this block is illustrated in Figure 8.3. This block takes the original process model and makes modifications to it according to the parameters "number of drifts" and "type of change". If the number of drifts is set to more than one, then all subsequent modifications (except the first one) will be applied to the already modified models (i.e., incremental drifts) instead of modifying the original model again. This results in a set of "modified" process models. Then, an event log is built by sampling traces from such set of "original" and "modified" models according to the duration distributions and sampling probabilities defined by the parameters.

Sampling probabilities determine the specific model from which a trace is sampled. For example, a sudden drift between models A and B will change the sampling probabilities of model A from 1 (before the drift) to 0 (after the drift) and of B in an inverse way. In the case of gradual drifts, such sample probabilities do not change so abruptly, but more gradually over time. The duration of a gradual drift is set by the parameter "duration of drift period". Within a gradual drift, the sampling probabilities can change over time in different ways,

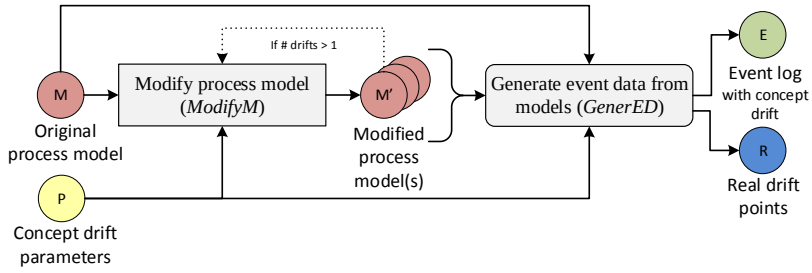


Figure 8.3: Inner composition of the “Generate event data with concept drift from models" block. A process model is modified a given number of times. Then, an event log is sampled from the resulting collection of models (original and modified) according to some parameters.

and are defined by the “transition function” parameter. An example of this is illustrated in Figure 8.4, where two gradual drifts are shown. The first (left) has a *linear* transition function for the sampling probabilities of models A and B i.e., the sampling probabilities of both models change linearly over time until traces are no longer sampled from model A (sampling probability = 0) and are only sampled from model B (sampling probability = 1). The second (right) has an *exponential* transition function for the sampling probabilities of models B and C i.e., the sampling probabilities of both models change exponentially over time until traces are no longer sampled from model B (sampling probability = 0) and are only sampled from model C (sampling probability = 1).

The “duration of stable period” parameter can be used to define how much “training” data can be used by the concept drift detection techniques. For example, short “stable periods” lead to little event data available for describing the (initial) stable states of the process, hence making it harder for techniques to detect drifts. This is also combined with the last two parameters (i.e., “time between cases” and “time between events”) to determine the *density* of event data over time. For example, the user can prefer to have many co-existing traces (i.e., a short “time between cases) that are long-running and spread over time (i.e., a long “time between events”). Note that these last two parameters also apply, in the case of gradual drifts, to the “duration of drift period”.

The results of this block are an event log that contains concept drift, and the

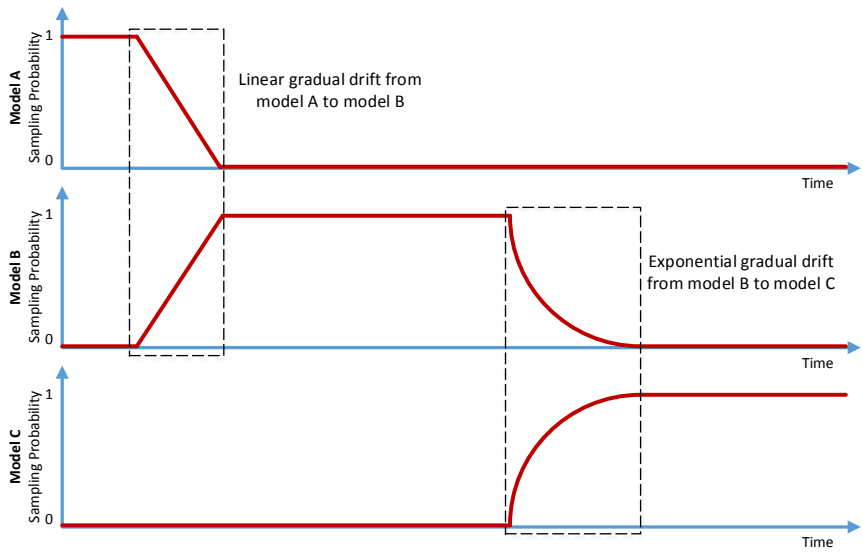


Figure 8.4: Example of linear and exponential transition functions for sampling probabilities of different models in gradual drifts.

set of points in time in which such drifts occurred i.e., the "real" drift points.

Analyze event data

This block represents the application of a concept drift detection technique to the event log with concept drift obtained from the previous block. Concept drift detection can be seen as a type of event log analysis: an event log is used as the input, and a set of "discovered" drift points in time is obtained as a result. These "discovered" drift points will be used as an input in the next block.

As claimed before, the framework reported in this chapter is extensible to incorporate new concept drift detection techniques, as long as they produce a set of points in time in which it detected drift. This is only possible because this block is not bound to a specific implementation.

Calculate metrics

The framework summarizes the performance of a concept drift detection technique by using three standard metrics adopted from the data mining and information retrieval domain: precision, recall and F-measure. In the case of concept drift [4], they are based on the following metrics:

- True Positives (TP): the number of **real** drifts that were **discovered** by the technique.
- False Positives (FP): the number of drifts **discovered** by the technique that **do not match** with any real drift.
- False Negatives (FN): the number of **real** drifts that were **not discovered** by the technique.

In this case, true negatives (i.e., points in time in which there is no "discovered" nor "real" drift) are not defined, because they are infinite.

Given the fact that time is a continuous variable, the "discovered" drift point in time will never exactly match the "real" drift. Therefore, an interval of correctness (i.e., *epsilon*) is used for this purpose. If the distance between a "discovered" drift point and a "real" drift point is less or equal than *epsilon*, then it is considered as a true positive.

It is important to note that a "discovered" drift can be associated to only one "real" drift and viceversa, in order to prevent techniques from benefitting by detecting the same drift many times. Also note that, in the case of gradual drifts, any "discovered" drift point detected within a drift period will not be considered as a false positive, since gradual drifts can also be seen as a sequence of "smaller" sudden drifts.

Figure 8.5 illustrates this through an example of a mapping between discovered drifts and real drifts. The discovered drift point d_1 is mapped to the sudden real drift r_1 because it is located within its interval of correctness ϵ. Even though the discovered drift d_2 is also located within the same epsilon, it is not mapped to r_1 because d_1 is. Hence, from d_1 and d_2, only the first is considered as a true positive. The second cannot be a false positive because it is not incorrect i.e., there is indeed a drift in that moment. Hence, in order to prevent techniques from discovering the same drift multiple times and benefitting in the process, it is not considered in the quality metrics. The discovered drift d_3 is considered as a false positive because it cannot be related to any real drift. The sudden real drift r_2 is considered as a false negative, because there is no discovered drift within its interval of correctness ϵ. The gradual drift that starts in r_3 and ends

in r_4 is mapped to the discovered drift d_4 and is considered as a true positive. The same gradual drift is discovered again in d_5. In fact, a gradual drift can be seen as a continuous sequence of drifts. Hence, any drift discovered during a gradual drift period is indeed correct. However, in this framework, d_5 (and also d_2) is not considered as a true positive in order to prevent techniques from discovering the same drift multiple times and benefitting from this.

Like in the previous chapter, we define the quality metrics for concept drift detection techniques based on the metrics defined above. The *precision* metric mentioned before refers to the percentage of "real" drifts that were correctly detected by the technique from all the "discovered" drifts.

$$\text{Precision} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Positives})}$$

Inversely, the *recall* metric refers to the percentage of "real" drifts that were correctly detected by the technique from all the "real" drifts in the event data.

$$\text{Recall} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Negatives})}$$

The framework uses the F_1 variation of the F-measure. This statistic refers to the harmonic average of the precision and recall metrics.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{(\text{Precision} + \text{Recall})}$$

Statistical analysis

The evaluation framework allows users to compare the performance of algorithms and to study the effect of control-flow characteristics on algorithm per-

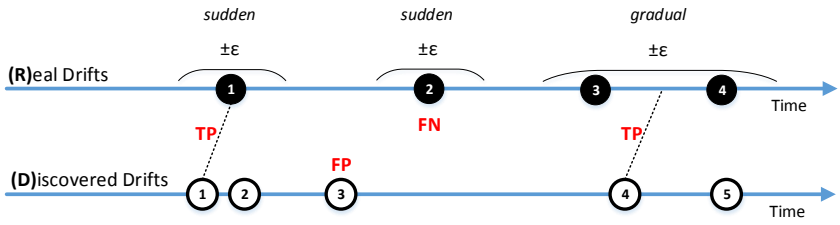


Figure 8.5: Mapping of discovered drifts and real drifts for calculating quality metrics.

formance. The statistical analysis based on the evaluation results depends on the objectives of the experiment and the corresponding hypotheses to be tested. Therefore, the framework does not incorporate specific statistical techniques, instead it can be used with a whole range of exploratory, descriptive and causal statistical techniques to test any hypothesis that can be expressed in terms of precision, recall, F_1 score, and characteristics of log and model. The authors believe that this will benefit the adoption of the framework for all types of evaluation studies, rather than serve a specific purpose.

8.3 Experiments

The framework presented in this chapter has been is implemented a *process mining workflow* (see Chapter 4). Therefore, other researchers can replicate experiments with little effort by just executing the workflow. In this way, our framework facilitates repeated concept drift detection evaluation, e.g., it becomes trivial to evaluate another set of algorithms or to assess the algorithm's performance with respect to other data characteristics (e.g. noise, control-flow patterns, etc.).

The framework was operationalized in RapidProM (presented in Chapter 4), which contains instantiations for all the blocks mentioned in Section 8.2.2. Figure 8.6 illustrates the implemented workflow.

The experiments were based on the following concept drift detection techniques: *ProDrift* [111] is used twice, each with a different option: event-stream-based (i.e., *ProDrift (event)*) and trace-stream-based (i.e., *ProDrift (trace)*), *ConceptDrift* [116] and *VariantFinder* i.e., the process variant detection technique presented in Chapter 6.

All the techniques were used with the default configuration. Naturally, the parameter configuration can have a huge impact on the technique's performance. However, the objective of this section does not relate to finding the best parameters of each algorithm, but to study how these techniques are affected by other variables e.g., the type and quantity of drifts.

For this purpose, we conducted an experiment that validates the usefulness of the proposed framework through a large-scale experiment consisting of a detailed empirical analysis of the concept drift detection techniques mentioned above in many different scenarios. Note that the experiment performed in this section is around ten times larger than the experiment presented in Section 7.3.2.

The implemented workflow that executes the experiments presented in this

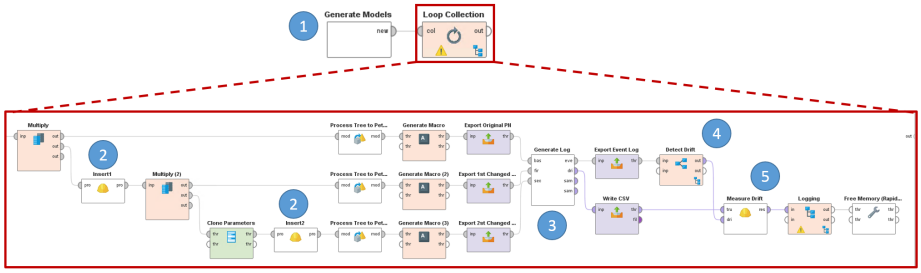


Figure 8.6: Concrete implementation of the framework into a RapidMiner workflow. In Step 1, a collection of models is generated from a population settings parameter. In Step 2, for each generated model, a set of two models modified with concept drift is created. In Step 3, the original and the two modified models are used to create an event log with concept drift according to the specified parameters. In Step 4, a concept drift detection technique is used to detect the points of drift. In Step 5, these are then compared with the original drift points. Finally, results are processed.

section (see Figure 8.6) is publicly available, and can be downloaded from: `https://www.dropbox.com/s/7gnwmp4pgoas1s2/ConceptDriftBenchmark.zip?dl=0`. The reader is encouraged to download and use the workflow to reproduce the experiment results or extend the framework to other concept drift detection algorithms or process characteristics. The remainder of this section describes the experimental setup and discusses the results obtained from it.

As mentioned earlier, the goal of this framework is to analyze and compare the accuracy of concept drift detection techniques to rediscover concept drifts based on a population of observed executions i.e., event logs. In the experiment, the population of event logs with concept drift is generated by varying several parameters, such as the probability of parallelism in the underlying model, the type of drift and change in the process, the duration of stable and drifting periods, the density of event data in such periods, and the interval of correctness (i.e., epsilon).

In this way, we can e.g., study the impact of parallelism in a process on the accuracy of concept drift detection techniques. The accuracy can also be evaluated in many different concept drift scenarios e.g., when parts of the process are *removed* in a *sudden drift*.

Therefore, the experimental design includes all the combinations of these

independent variables. The parameter combinations considered in this experiment for such variables are summarized in Table 8.2.

Table 8.2: Parameter combinations considered in the experiment.

Parameter Type	Parameter	Considered Values
Technique	Approach	ProDrift (events) [111], ProDrift (runs) [111], VariantFinder (Chapter 6), ConceptDrift [116]
Model size (# act)	min mode max	10 20 30
Probability of basic c-f pattern	sequence (s) exclusive choice (e) parallelism (p) inclusive choice (i) loop (l)	$(1-p)*0.57$ $(1-p)*0.43$ {0.0, 0.1, 0.2, 0.3} 0 0
Probability of advanced c-f pattern	silent transitions duplicate activities long-term dependency infrequent paths	0 0 0 false
Model changes	Number of drifts Type of change	2 {add, remove, swap fragments}
Drift properties	Type of drift Duration of stable period Duration of drift period (*gradual drift*) Transition function (*gradual drift*)	{sudden, gradual} exponential(1 year) exponential({0.5, 1, 1.5} months) {linear, exponential}
Event log density	Time between cases Time between events	exponential({0.5, 1, 1.5} days) exponential(1 day)
Accuracy	Epsilon	{1, 25, 50} days

In total, 3024 possible parameter combinations are included in the experiment: 432 combinations for *sudden* drift (i.e., 4 concept drift detection techniques $\times$ 4 probabilities of parallelism $\times$ 3 types of changes $\times$ 3 different epsilons $\times$ 3 distributions that define the time between cases) and 2592 combinations for *gradual* drift (i.e., 4 concept drift detection techniques $\times$ 4 probabilities of parallelism $\times$ 3 types of changes $\times$ 3 different epsilons $\times$ 3 distributions that define the time between cases $\times$ 3 durations of a drift period $\times$ 2 drift transition functions). Note that the last two "Drift properties" defined in Table 8.2 (i.e., "duration of drift period" and "transition function") apply only in the case of gradual drifts, hence, there is a different number of parameter combinations for sudden and gradual drifts.

Given the high number of parameter combinations, other process characteristics (defined in Section 8.2.2) were not taken into account in this analysis. Concretely, the probability of non-exclusive choice (OR) and of loops are set to zero and, hence, these two constructs do not occur. The size of models is de-

fined by a fixed triangular distribution from 10 to 30 activities, with a mode of 20 activities per model. The ratio of the probability of sequence and exclusive choice set and kept fixed to 46/35 (i.e., the same as the "fixed" values used in the previous chapter) and after a probability of parallelism is set to p, the remainder $1 - p$ is distributed among the probabilities of sequence and exclusive choice using this ratio. These values have been determined after analyzing their frequencies in the large collections of models, as reported in [98]. The reader is referred to Section 7.3.1 for a detailed explanation of this choice.

We also fixed some drift-related characteristics (defined in Section 8.2.2) in order to reduce the number of parameter combinations. Concretely, we defined a fixed "number of drifts" to 2 concept drifts per event log, a fixed "duration of stable periods" (where no drifts occur) defined as an exponential function with a mean of 1 year, and a fixed "time between events" defined as an exponential function with a mean of 1 day.

For each parameter combination a random sample of 68 process models is drawn. The sample size of 68 models allows us to study the interactions of the eight variables described before using a fixed effects ANOVA analysis [48] with significance level $\alpha = 0.05$ and power $1 - \beta = 0.99$. This power indicates the probability to detect a significant effect when two concept drift detection algorithms actually differ by a small difference. In total (i.e., sum of all combinations), 205,632 (i.e., 3024×68) different process models were generated. For each of the obtained process models, an event log with concept drift is generated (See Section 7.2.2). The exact size of an event log is not defined *a priori*, as it will depend on other parameters defined above, such as the "time between cases" and the "duration of stable period".

The effects and interactions of the eight variables defined before can be analyzed using one-way ANOVA analysis if the assumptions of homogeneity of variances and normality of the dependent variable hold [48]. However, similarly to Chapter 7, both assumptions were also violated for every dependent variable, i.e. F_1, recall and precision. Therefore, the non-parametric *Kruskall-Wallis* test (KW) [136] was applied instead. Finally, the Jonckheere test [136] is used to test for a significant trend in the data. See Section 7.3.1 for a detailed explanation of both tests.

The remainder of this section reports on the analysis of the experimental results, where we measure the effect of each variable defined in Table 8.2 on recall, precision and F_1 score. The remainder of this section is then organized as follows. Section 8.3.1 describes the effect of the concept drift detection techniques on the accuracy of drift point detection. Section 8.3.2 describes the effect of parallelism on the accuracy of drift point detection. Section 8.3.3 describes

the effect of the type of drift (i.e., sudden or gradual) on the accuracy of drift point detection. Section 8.3.4 describes the effect of the type of change in a process (i.e., add, remove or swap fragments of the process) on the accuracy of drift point detection. Section 8.3.5 describes the effect of the time between cases on the accuracy of drift point detection. Finally, Section 8.3.6 describes the effect of the duration and transition functions of gradual drift on the accuracy of drift point detection.

8.3.1 The Effect of Concept Drift Detection Technique

The first goal of this experiment is to learn the effect of the concept drift detection technique (i.e., the "approach" variable) on each of the dependent variables: recall, precision and F_1 score. In other words, we want to analyze whether some techniques have better overall quality metrics than others. All other independent variables are part of the error term.

For this purpose, we apply the KW method, to test whether the average rank differs between the four concept drift detection techniques (i.e., samples). In this case we ranked all the 205,632 values for recall, precision and F_1 scores ignoring sample membership (i.e., concept drift detection technique). The highest value for recall, precision and F_1 score gets rank 1 (lowest rank), while the lowest absolute value gets rank 205,632 (highest rank). Then we computed the average ranking per concept drift detection technique. A lower average ranking means better performance. The ranking summary is shown in Table 8.3.

Table 8.3: Average ranks of concept drift detection techniques (n = 205,632). Each cell indicates the average ranking for a specific performance dimension (row header) and for a specific concept drift detection technique (column header).

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	115,363.4	109,061.4	121,185.4	65,655.8
Precision	107,123.1	114,104.7	109,996.1	80,042.1
F_1 score	109,259.7	114,162.8	113,295.4	74,548.1

Based on the average rankings in Table 8.3, the order suggested between concept drift detection techniques is: *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (trace)* $\prec$ *ProDrift (event)* for recall, precision and F_1 scores. It means that the *VariantFinder* is the most accurate for finding drift points in terms of recall, precision and F_1 scores. The *ConceptDrift* technique outperforms both versions of *ProDrift*. The results of the KW test confirm that the all differences in average rankings between the four concept drift detection techniques are statistically

significant (significance level $\alpha = 0.05$). Moreover, the multiple comparison post-hoc test (cf. supra) also confirms the statistical significance of the differences between techniques. Table 8.4 shows a summary of the statistical test results for the F_1 scores.

Table 8.4: Results of the statistical tests to study the effect of concept drift detection technique on F_1 scores for detecting *sudden* drift.

Kruskall-Wallis rank sum test		
KW $\chi^2 = 19602$	degrees of freedom $= 3$	p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)		
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
ConceptDrift - ProDrift (event)	4903.02	976.82	True
ConceptDrift - ProDrift (trace)	4035.66	976.82	True
ConceptDrift - VariantFinder	34711.64	976.82	True
ProDrift (event) - ProDrift (trace)	867.36	976.82	False
ProDrift (event) - VariantFinder	39614.66	976.82	True
ProDrift (trace) - VariantFinder	38747.30	976.82	True

However, these results do not suggest which one is the "best" overall technique. It is important to note that the *VariantFinder* approach only works in an *offline* setting, while the other approaches also work in an *online* setting. Also, it is important to note that these techniques were used with default parameters. Naturally, a good selection of parameter values can have an huge impact on accuracy. However, hyperparameter analysis is outside the scope of this experiment.

Computation time should also be considered when choosing a technique. Figure 8.7 shows the average computation times for each technique included in this experiment. We can observe a trade-off between F_1 score and computation time: the *VariantFinder* approach is the most accurate yet the slowest technique, while the *ProDrift (trace)* approach is not as accurate yet is the fastest technique by far.

Hence, it is recommended to carefully consider these factors when choose a technique, based on the analysis conditions, purpose and desired accuracy.

8.3.2 The Effect of Parallelism

This analysis investigates how the accuracy of each concept drift detection technique (in terms of precision, recall and F_1 score) is influenced by the average

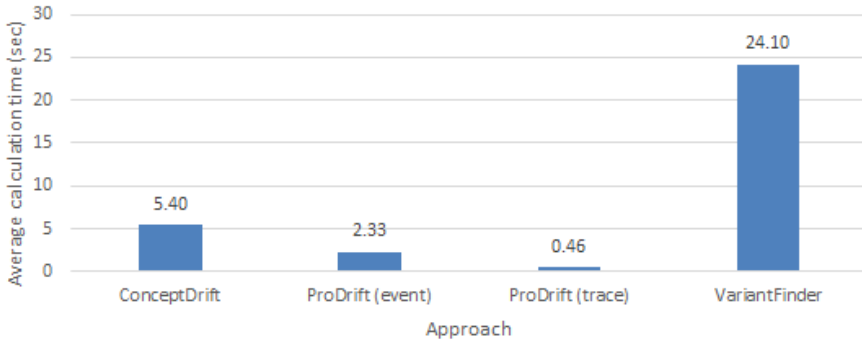


Figure 8.7: Average calculation time for each concept drift detection technique.

percentage of "parallelism" constructs in the process models (i.e., the "probability of parallelism" variable). The effect of the other independent variables is a part of the error term.

Figure 8.8 illustrates the average F_1 scores for all the concept drift detection techniques over different probabilities of parallelism.

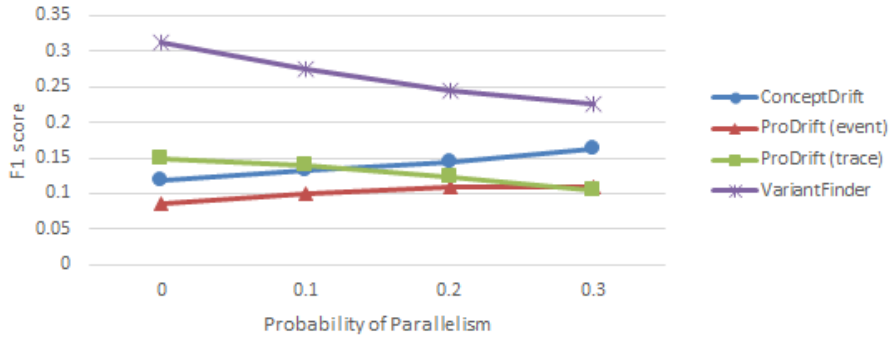


Figure 8.8: F_1 scores for concept drift detection for different probabilities of parallelism in the process.

This graph indicates a negative trend (i.e., the probability of parallelism has a negative effect on F_1 scores) for the *VariantFinder* and *ProDrift (trace)* approaches, while it indicates a positive trend for the *ConceptDrift* and *ProDrift*

(event) approaches.

To determine whether such trends are statistically significant, an in-depth analysis is performed. First, the sample is divided into subsets grouped by concept drift detection technique. Each subset then contains 51,480 values for recall, precision and F_1 score values ignoring sample membership (i.e., concept drift detection technique). As such, the variation in accuracy associated with the approach is isolated. Then, similar to the analysis above, the KW test is applied to compare the average rankings of recall, precision and F_1 score values.

Table 8.5 contains one subtable for each concept drift detection technique with the average ranks for all three metrics by the probability of parallelism.

For the *ConceptDrift* approach, the data (shown in Table 8.5a) seems to suggest that as the probability of parallelism increases, its accuracy for detecting drift points increases as well in terms of recall, precision and F_1 score. To test this impression statistically, we will rely on the KW and Jonckheere tests. The results of these statistical tests and the multiple pair-wise comparison between different probabilities of parallelism for the *ConceptDrift* approach are presented in Table 8.6. Both tests show that there is indeed a statistically significant positive trend in the F_1 scores as the probability of parallelism increases. The pairwise comparison of each probability of parallelism is very clear about how this trend looks like for F_1 scores, with all comparisons statistically significant.

Similarly, for the *ProDrift (event)* approach, the data (shown in Table 8.5b) also suggest a positive correlation between the probability of parallelism and the quality of the drift detection. To test this impression statistically, we will again rely on the KW and Jonckheere tests. The results of these statistical tests and the multiple pair-wise comparison between different probabilities of parallelism for the *ProDrift (event)* approach are presented in Table 8.7. Both tests show that there is also a strong trend in the relative quality of concept drift detection as the probability of parallelism increases. A pairwise comparison of each probability of parallelism confirms that all comparisons are statistically significant.

Unlike the approaches described above, for the *ProDrift (trace)* approach, the data (shown in Table 8.5c) suggest a negative correlation between the probability of parallelism and the recall, precision and F_1 scores. This means that the quality of detection seems to degrade for increasing levels of parallelism. To test this impression statistically, we will again rely on the KW and Jonckheere tests. The results of these statistical tests and the multiple pair-wise comparison between different probabilities of parallelism for the *ProDrift (trace)* approach are presented in Table 8.8. Both tests show that there is indeed evidence of a negative trend in the relative quality of concept drift detection as the probability of parallelism increases. A pairwise comparison of each probability of parallelism

Table 8.5: Average ranks of concept drift detection techniques per probability of parallelism in terms of precision, recall and F_1 scores.

(a) ConceptDrift [116] (n = 51,480)

Prob. Parallelism	0	0.1	0.2	0.3
Recall	27,022.82	26,149.63	25,379.68	24,265.87
Precision	26,826.53	26,115.59	25,420.84	24,455.05
F_1 **score**	26,935.68	26,150.53	25,406.23	24,325.56

(b) ProDrift (event) [111] (n = 51,480)

Prob. Parallelism	0	0.1	0.2	0.3
Recall	28,587.52	26,582.24	24,597.07	23,051.17
Precision	27,776.88	26,120.49	24,835.70	24,084.93
F_1 **score**	27,813.16	26,157.47	24,811.47	24,035.90

(c) ProDrift (trace) [111] (n = 51,480)

Prob. Parallelism	0	0.1	0.2	0.3
Recall	24,592.10	25,053.44	26,077.07	27,095.39
Precision	24,678.22	25,044.76	26,039.83	27,055.20
F_1 **score**	24,634.35	25,055.18	26,059.91	27,068.56

(d) VariantFinder [Ch. 6] (n = 51,480)

Prob. Parallelism	0	0.1	0.2	0.3
Recall	28,671.65	26,285.22	24,588.50	23,272.63
Precision	22,861.34	25,149.99	26,918.15	27,888.52
F_1 **score**	23,711.50	25,187.16	26,551.51	27,367.84

shows that most comparisons are statistically significant, except for the range [0.0 - 0.1]. This mean that, even though there is a trend, it is weaker than the trends found for the other approaches described above.

Finally, for the *VariantFinder* approach, the data (shown in Table 8.5d) suggests that increasing probabilities of parallelism have a positive effect on recall, but a negative effect on precision and F_1 scores. To test this impression statistically, we will again rely on the KW and Jonckheere tests. The results of these

Table 8.6: Results of the statistical tests to study the effect of parallelism on F_1 scores for the *ConceptDrift* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 303.38$ degrees of freedom $= 3$ p-value $= 2.2e^{-16}$			
Jonckheere-Terpstra test			
JT $= 523103696$ p-value $= 0.0002$			
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.1	785.15	488.41	True
0.0-0.2	1529.45	488.41	True
0.0-0.3	2610.12	488.41	True
0.1-0.2	744.30	488.41	True
0.1-0.3	1824.97	488.41	True
0.2-0.3	1080.67	488.41	True

Table 8.7: Results of the statistical tests to study the effect of parallelism on F_1 scores for the *ProDrift (event)* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 661.5$ degrees of freedom $= 3$ p-value $< 2.2e^{-16}$			
Jonckheere-Terpstra test			
JT $= 535992606$ p-value $= 0.0002$			
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.1	1655.69	488.41	True
0.0-0.2	3001.69	488.41	True
0.0-0.3	3777.26	488.41	True
0.1-0.2	1346.00	488.41	True
0.1-0.3	2121.57	488.41	True
0.2-0.3	775.57	488.41	True

statistical tests and the multiple pair-wise comparison between different probabilities of parallelism for the *VariantFinder* approach is presented in Table 8.9. Both tests show strong evidence of a negative trend in the relative quality in terms of F_1 score as the probability of parallelism increases. A pairwise comparison of each probability of parallelism only confirms this strong negative trend: all comparisons are statistically significant.

Table 8.8: Results of the statistical tests to study the effect of parallelism on F_1 scores for the *ProDrift (trace)* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 332.17$	degrees of freedom $= 3$	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT $= 468834226$		p-value $= 0.0002$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.1	420.82	488.41	False
0.0-0.2	1425.56	488.41	True
0.0-0.3	2434.20	488.41	True
0.1-0.2	1004.73	488.41	True
0.1-0.3	2013.38	488.41	True
0.2-0.3	1008.64	488.41	True

Table 8.9: Results of the statistical tests to study the effect of parallelism on F_1 scores for the *VariantFinder* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 463$	degrees of freedom $= 3$	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT $= 455492769$		p-value $= 0.0002$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.1	1475.65	488.41	True
0.0-0.2	2840.01	488.41	True
0.0-0.3	3656.33	488.41	True
0.1-0.2	1364.35	488.41	True
0.1-0.3	2180.68	488.41	True
0.2-0.3	816.32	488.41	True

In summary, we can note that increasing levels of parallelism in a process produces a positive effect in the accuracy of the *ConceptDrift* and *ProDrift (event)* approaches, and a negative effect in the accuracy of the *ProDrift (trace)* and *VariantFinder* approaches.

8.3.3 The Effect of Type of Drift

This analysis tests whether the "type of drift" (i.e., sudden or gradual) has an impact on the average ranking of the four concept drift detection techniques for recall, precision and F_1 scores. The effect of all the other independent variables is part of the error term. Firstly, the sample is split into two subsets: experiments with *sudden drift* (29,376 values for recall, precision and F_1 score) and experiments with *gradual drift* (176,256 values for recall, precision and F_1 score). This division is called *blocking* (see Section 7.2) which is done to isolate the variation in recall, precision and F_1 scores attributable to the type of drift. Secondly, the KW test is applied to each subset. Table 8.10 shows the average rankings per approach, grouped by the type of drift.

Table 8.10: Average ranks of concept drift detection techniques in terms of precision, recall and F_1 scores for different types of drift.

(a) Sudden drift (n = 29,376)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	14,821.43	15,613.23	17,757.42	10,561.92
Precision	13,826.38	16,198.78	17,013.38	11,715.46
F_1 **score**	13,846.78	16,180.57	17,075.99	11,650.66

(b) Gradual drift (n = 176,256)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	100,826.81	93,360.57	103,252.97	55,073.65
Precision	93,275.95	97,953.11	92,966.28	68,318.66
F_1 **score**	95,775.35	98,003.42	96,126.46	62,608.77

Unlike the analysis presented in Section 8.3.1, the order suggested by this analysis differs for sudden and gradual drift.

For sudden drift, the order suggested is: *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (event)* $\prec$ *ProDrift (trace)*. The KW test and multiple comparison post-hoc tests indicated that all the approaches have statistically-significant differences with all the other approaches.

For gradual drift, the order suggested differs from sudden drift, and is: *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (trace)* $\prec$ *ProDrift (event)*. Unlike the case of sudden drift, multiple comparison post-hoc test show that, for gradual drift, the *ConceptDrift* and *ProDrift (trace)* approaches do not have a statistically-significant difference in terms of F_1 scores. Table 8.11 shows a summary of the statistical post-hoc test results for the F_1 scores for gradual drift.

Table 8.11: Results of the statistical tests to study the effect of concept drift detection technique on F_1 scores for *gradual* drift.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 18264$	degrees of freedom $= 3$		p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
ConceptDrift - ProDrift (event)	2228.07	904.36	True
ConceptDrift - ProDrift (trace)	351.11	904.36	False
ConceptDrift - VariantFinder	33166.57	904.36	True
ProDrift (event) - ProDrift (trace)	1876.96	904.36	True
ProDrift (event) - VariantFinder	35394.64	904.36	True
ProDrift (trace) - VariantFinder	33517.68	904.36	True

Note that only the difference between the *ConceptDrift* and *ProDrift (trace)* approaches in the presence of gradual drift is not statistically significant for F_1 scores. Therefore, one cannot accept the assumption that the type of drift does not influence concept drift detection techniques.

8.3.4 The Effect of Type of Change

This analysis tests whether the "type of change" variable (i.e., add fragment, remove fragment, swap fragment) has an impact on the average ranking of the four concept drift detection techniques for recall, precision and F_1 scores. The effect of all the other variables are a part of the error term. Firstly, the sample is split into three subsets depending on the type of change. Each subset contains 68,544 values for recall, precision and F_1 scores. Secondly, the KW test is applied to each subset. Table 8.12 contains the average rankings per concept drift detection technique grouped by the type of change included in the experiment.

These rankings (particularly for F_1 score) indicate that for each type of change, there is a different ordering:

"add fragment": *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (event)* $\prec$ *ProDrift (trace)*

"remove fragment": *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (trace)* $\prec$ *ProDrift (event)*

Table 8.12: Average ranks per concept drift detection technique in terms of F_1 scores.

(a) Type of change = "add fragment" (n = 68,544)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	37,339.11	36,224.63	41,382.03	22,144.23
Precision	34,628.75	38,211.44	38,289.51	25,960.30
F_1 **score**	35,379.04	38,276.27	39,332.26	24,102.43

(b) Type of change = "remove fragment" (n = 68,544)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	36,103.29	37,032.86	39,500.70	24,453.15
Precision	33,768.64	39,283.46	36,520.79	27,517.11
F_1 **score**	34,343.68	39,155.18	37,443.88	26,147.27

(c) Type of change = "swap fragment" (n = 68,544)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	41,821.45	35,899.60	40,390.44	18,978.52
Precision	38,841.20	36,456.73	35,096.23	26,695.84
F_1 **score**	39,626.78	36,612.76	36,430.59	24,419.87

"swap fragment": *VariantFinder* $\prec$ *ProDrift (trace)* $\prec$ *ProDrift (event)* $\prec$ *ConceptDrift*

This leads to the assumption that the concept drift techniques are heavily influenced by the type of change. For the types of change "add fragment" and "swap fragment", the KW and multiple comparison post-hoc tests show that all the techniques have statistically-significant differences with all the other techniques. However, this is not the case for the type of change "remove fragment", as the multiple comparison post-hoc test shows that the *ProDrift (event)* and *ProDrift (trace)* techniques do not have a statistically-significant difference in terms of F_1 score. Table 8.13 shows a summary of the statistical post-hoc test results for the F_1 scores for the type of change "remove fragment".

Note that only the difference between the *ProDrift (event)* and *ProDrift (trace)* techniques for the type of change "remove fragment" is not statistically significant for F_1 scores. However, all the other comparisons are highly significant. We can conclude that the type of change has (in general) a considerable impact on the accuracy of concept drift detection techniques.

Table 8.13: Results of the statistical tests to study the effect of the concept drift detection technique on F_1 scores for the type of change "remove fragment".

Kruskall-Wallis rank sum test			
KW $\chi^2 = 7080.3$	degrees of freedom $= 3$	p-value $< 2.2e^{-16}$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
ConceptDrift - ProDrift (event)	3014.02	563.97	True
ConceptDrift - ProDrift (trace)	3196.19	563.97	True
ConceptDrift - VariantFinder	15206.90	563.97	True
ProDrift (event) - ProDrift (trace)	182.16	563.97	False
ProDrift (event) - VariantFinder	12192.88	563.97	True
ProDrift (trace) - VariantFinder	12010.71	563.97	True

8.3.5 The Effect of Time Between Cases

This analysis investigates how the accuracy of each concept drift detection technique (in terms of precision, recall and F_1 score) is influenced by the "time between cases" variable. The time between cases can influence the "density" of cases in time: a higher value will likely generate event logs with traces more spaced between each other in terms of time, and a lower value will provoke traces to be closer to each other in terms of time, hence, their overlap in a timescale increases. The effect of the other independent variables is part of the error term.

Figure 8.9 illustrates the average F_1 scores for all the concept drift detection techniques over different times between cases.

This graph indicates a general negative trend (i.e., a "longer" time between cases has a negative effect on F_1 scores) for the *ProDrift (trace)*, *ConceptDrift* and *ProDrift (event)* approaches, while it seems not to affect the *VariantFinder* approach.

To determine whether such trends are statistically significant, the sample is divided into subsets grouped by concept drift detection technique. Each subset then contains 51,480 values for recall, precision and F_1 score values ignoring sample membership (i.e., concept drift detection technique). As such, the variation in accuracy associated with the approach is isolated. Then, similar to the analysis described in Section 8.3.2, the KW test is applied to compare the average rankings of recall, precision and F_1 score values

Table 8.14 contains one subtable for each concept drift detection technique

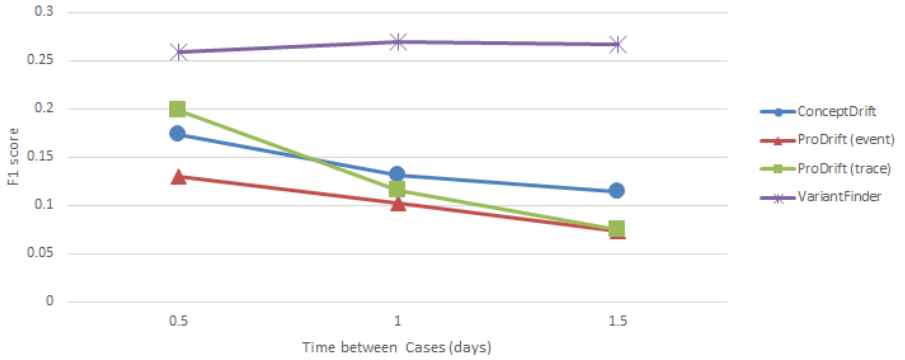


Figure 8.9: F_1 scores for concept drift detection techniques for different values of "time between cases".

with the average ranks for all three metrics by the different values of time between cases.

The data shown in Table 8.14a seems to confirm the initial belief that, for the *ProDrift (trace)*, *ConceptDrift* and *ProDrift (event)* approaches, when the "time between cases" increases, their accuracy for detecting drift points decreases in terms of recall, precision and F_1 score. To test this belief statistically, we will rely on the KW and Jonckheere tests. The results of these statistical tests and the multiple pair-wise comparison between different times between cases indicated that there is indeed a statistically significant negative trend in the F_1 scores as the time between cases increases. The pairwise comparison of each value of the "time between cases" variable is determinant about how this trend looks like for F_1 scores, with all comparisons statistically significant.

Unlike the analysis presented above, the data shown in Table 8.14a seems to confirm that the accuracy of the *VariantFinder* approach is not affected by the time between cases. To statistically validate this claim, we applied the KW and Jonckheere tests. The results of these statistical tests are shown in Table 8.15. The KW test shows that there are statistically significant differences in terms of F_1 scores for different "time between cases". However, the Jonckheere test indicates that there is no statistically-significant trend (either positive or negative) in the accuracy of the *VariantFinder* approach.

In summary, we can note that increasing "time between cases" (i.e., event logs with more overlap between cases) has an negative impact in the accuracy of all the approaches included in the experiment. Moreover, only for the *Vari-*

Table 8.14: Average ranks of concept drift detection techniques for different times between cases in terms of precision, recall and F_1 scores.

(a) ConceptDrift [116] (n = 51,480)

Time between Cases (days)	0.5	1	1.5
Recall	23,929.91	26,154.42	27,029.17
Precision	23,717.08	26,189.75	27,206.67
F_1 **score**	23,740.31	26,194.51	27,178.68

(b) ProDrift (event) [111] (n = 51,480)

Time between Cases (days)	0.5	1	1.5
Recall	21,456.37	26,215.38	29,441.74
Precision	22,578.75	25,829.75	28,704.99
F_1 **score**	22,480.16	25,862.37	28,770.97

(c) ProDrift (trace) [111] (n = 51,480)

Time between Cases (days)	0.5	1	1.5
Recall	22,253.65	26,311.06	28,548.79
Precision	22,673.18	26,177.63	28,262.69
F_1 **score**	22,317.18	26,308.03	28,488.29

(d) VariantFinder [Ch.6] (n = 51,480)

Time between Cases (days)	0.5	1	1.5
Recall	24,442.34	25,576.55	27,094.61
Precision	25,982.54	25,519.01	25,611.95
F_1 **score**	25,886.24	25,466.45	25,760.81

antFinder approach there was no negative trend. It is important to note that, from all the approaches, *VariantFinder* is the only one that does not use *sliding windows*, but uses all the data available at once.

Table 8.15: Results of the statistical tests to study the effect of time between cases on F_1 scores for the *VariantFinder* approach.

Kruskall-Wallis rank sum test		
KW $\chi^2 = 7.4227$	degrees of freedom $= 2$	p-value $= 0.02445$
Jonckheere-Terpstra test		
JT $= 441959079$		p-value $= 0.4132$

8.3.6 The Effect of the Duration and Transition Functions of Gradual Drifts

This analysis focuses on the effect of two variables related to *gradual drift* ("Duration of Drift Period" and "Drift Transition Function") on the accuracy of each concept drift detection technique in terms of precision, recall and F_1 score.

The "Duration of Drift Period" variable influences amount of cases that are sampled during a period of gradual drift: a higher value will provide more observations (i.e., cases) of that drift period that can be used by the approaches to detect concept drift. The effect of the other independent variables is a part of the error term.

First, we analyze the effect of the "Duration of Drift Period" variable on the accuracy of approaches. Figure 8.10 illustrates the average F_1 scores for all the concept drift detection techniques over different duration of drift periods.

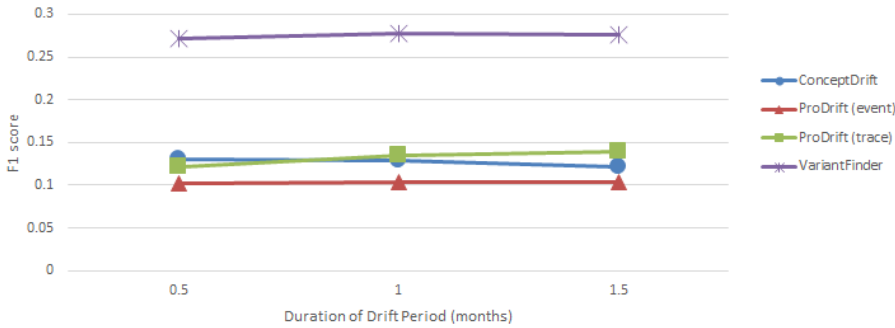


Figure 8.10: F_1 scores for concept drift detection techniques for different values of "duration of drift period".

This graph indicates that the duration of drift periods seems not to affect

the accuracy of the approaches. To validate this claim, the sample related to *gradual drift* (176,256 values for recall, precision and F_1 score) is divided into subsets grouped by concept drift detection technique. Each subset then contains 44,064 values for recall, precision and F_1 score values ignoring sample membership (i.e., concept drift detection technique). As such, the variation in accuracy associated with the approach is isolated.

Table 8.16 contains one subtable for each concept drift detection technique with the average ranks for all three accuracy variables by different duration of drift periods.

The data shown in Table 8.16a seems to indicate that the accuracy of the *ConceptDrift* approach is negatively affected by increasing durations of drift periods. To statistically validate this claim, we applied the KW and Jonckheere tests. The results of these statistical tests are shown in Table 8.18.

To the contrary, the data shown in Table 8.16b seems to indicate that the accuracy of the *ProDrift (event)* (in terms of F_1 score) is not affected by increasing durations of drift periods. To statistically validate this claim, we applied the KW test, which indicated that there are no significant differences in F_1 scores over increasing durations of drift periods (p-value = 0.2653). Therefore, we confirm that the duration of drift periods does not affect the accuracy of the *ProDrift (event)* approach.

For the *ProDrift (trace)* approach, the data shown in Table 8.16a seems to indicate that the accuracy of the approach is positively affected by increasing durations of drift periods. To statistically validate this claim, we applied the KW, Jonckheere and pairwise comparison post-hoc tests. The results of these statistical tests are shown in Table 8.18.

The test results confirm that there is indeed a statistically-significant positive trend on the accuracy of the approach over increasing durations of drift periods. Only in the range between 1 and 1.5 months of duration of drift periods there is no statistically-significant difference.

Finally, for the *VariantFinder* approach, the data shown in Table 8.16b seems to indicate that the accuracy (in terms of F_1 score) is not affected by increasing durations of drift periods. To statistically validate this claim, we applied the KW test, which indicated that there are no significant differences in F_1 scores over increasing durations of drift periods (p-value = 0.3437). Therefore, we confirm that the duration of drift periods does not affect the accuracy of the *VariantFinder* approach.

The remainder of this section describes the analysis performed to study the effect of the "drift transition function" variable in the accuracy (in terms of recall, precision and F_1 scores) of the four concept drift detection approaches

Table 8.16: Average ranks of concept drift detection techniques for different durations of drift periods in terms of precision, recall and F_1 scores.

(a) ConceptDrift [116] (n = 44,064)

Duration of Drift Period (months)	0.5	1	1.5
Recall	21,829.07	21,977.09	22,291.34
Precision	21,820.46	21,975.55	22,301.49
F_1 **score**	21,825.41	21,972.06	22,300.03

(b) ProDrift (event) [111] (n = 44,064)

Duration of Drift Period (months)	0.5	1	1.5
Recall	22,125.66	21,983.11	21,988.72
Precision	22,160.51	21,991.71	21,945.28
F_1 **score**	22,152.41	21,977.03	21,968.06

(c) ProDrift (trace) [111] (n = 44,064)

Duration of Drift Period (months)	0.5	1	1.5
Recall	22,532.43	21,899.29	21,665.78
Precision	22,526.32	21,926.06	21,645.12
F_1 **score**	22,537.43	21,907.28	21,652.79

(d) VariantFinder [Ch.6] (n = 44,064)

Duration of Drift Period (months)	0.5	1	1.5
Recall	21,684.05	21,991.31	22,422.15
Precision	22,167.39	21,943.37	21,986.75
F_1 **score**	22,136.37	21,922.27	22,038.87

included in the experiment. The "Drift Transition Function" variable defines the function that shapes the change of sampling probabilities during a drift period. In this experiment, we considered two different transition functions: linear (i.e., the sampling probabilities change *linearly* during the drift period) and exponential (i.e., the sampling probabilities have large changes in the beginning of the drift period, and then the changes become smaller by the end of the drift period). The effect of all the other independent variables is part of the error term.

Table 8.17: Results of the statistical tests to study the effect of the duration of drift periods on F_1 scores for the *ConceptDrift* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 15.202$	degrees of freedom $= 2$	p-value $= 0.0005$	
Jonckheere-Terpstra test			
JT $= 318960882$		p-value $= 0.0004$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.5-0.1	146.64	355.34	False
0.5-1.5	474.62	355.34	True
1.0-1.5	327.97	355.34	False

Table 8.18: Results of the statistical tests to study the effect of the duration of drift periods on F_1 scores for the *ProDrift (trace)* approach.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 57.839$	degrees of freedom $= 2$	p-value $= 2.757e^{-13}$	
Jonckheere-Terpstra test			
JT $= 332266195$		p-value $= 0.0004$	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.5-0.1	630.14	355.34	True
0.5-1.5	884.63	355.34	True
1.0-1.5	254.48	355.34	False

Firstly, the sample is split into two subsets: observations with *linear* and *exponential* drift transition function, each with 88,128 values for recall, precision and F_1 score. Table 8.19 shows the average rankings per approach, grouped by drift transition function.

The data suggests that in both cases, the ordering of approaches in terms of their average F_1 score rank is the same: *VariantFinder* $\prec$ *ConceptDrift* $\prec$ *ProDrift (trace)* $\prec$ *ProDrift (event)*. We confirmed this claim by applying the KW and multiple pairwise post-hoc comparison tests. Table 8.20 shows the test results for linear drift transition functions. Table 8.21 shows the test results for exponential drift transition functions.

These tests indicate the exact same statistical differences between approaches,

Table 8.19: Average ranks of concept drift detection techniques in terms of precision, recall and F_1 scores for different drift transition function.

(a) Linear Drift Transition Function (n = 88,128)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	50,349.27	46,684.71	51,619.53	27,604.50
Precision	46,586.44	48,914.36	46,562.63	34,194.57
F_1 **score**	47,839.87	48,956.02	48,116.68	31,345.44

(b) Exponential Drift Transition Function (n = 88,128)

	ConceptDrift [116]	**ProDrift (event)** [111]	**ProDrift (trace)** [111]	**VariantFinder** [Ch.6]
Recall	50,478.06	46,675.97	51,634.18	27,469.78
Precision	46,690.04	49,039.21	46,403.71	34,125.04
F_1 **score**	47,936.07	49,047.98	48,009.71	31,264.24

Table 8.20: Results of the statistical tests to study the effect of concept drift detection technique on F_1 scores for linear drift transition functions.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 9096.4$		degrees of freedom $= 3$	p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons		*Observed diff.*	*Critical diff.*	*Significant diff.*
ConceptDrift - ProDrift (event)		1116.14	639.48	True
ConceptDrift - ProDrift (trace)		276.80	639.48	False
ConceptDrift - VariantFinder		16494.43	639.48	True
ProDrift (event) - ProDrift (trace)		839.34	639.48	True
ProDrift (event) - VariantFinder		17610.58	639.48	True
ProDrift (trace) - VariantFinder		16771.24	639.48	True

regardless of the drift transition function used: only the difference between the *ConceptDrift* and *ProDrift (trace)* approaches is not statistically-significant.

Therefore, we can confirm that the drift transition function has no significant effect in the accuracy (in terms of F_1 score) of the four concept drift detection approaches included in the experiment.

Table 8.21: Results of the statistical tests to study the effect of concept drift detection technique on F_1 scores for exponential drift transition functions.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 9168.2$	degrees of freedom $= 3$		p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
ConceptDrift - ProDrift (event)	1111.90	639.48	True
ConceptDrift - ProDrift (trace)	73.64	639.48	False
ConceptDrift - VariantFinder	16671.83	639.48	True
ProDrift (event) - ProDrift (trace)	1038.26	639.48	True
ProDrift (event) - VariantFinder	17783.74	639.48	True
ProDrift (trace) - VariantFinder	16745.47	639.48	True

8.4 Conclusions

This chapter presented a novel evaluation framework for benchmarking concept drift detection techniques. The framework allows researchers to benchmark different approaches as well as to perform a sensitivity analysis to evaluate whether certain process and drift characteristics have an impact on the accuracy drift point detection. It allows to generalize the evaluation results to a user specified model population, also taking into account several properties that characterize concept drift in processes. Finally, the design of the framework as a process mining workflow enables automating, sharing and extending evaluation experiments.

The framework has been validated by conducting an extensive experiment involving four concept drift detection approaches that detect drift points in time and seven other variables related to process characteristics and concept drift characteristics. The experiment has shown the usefulness and flexibility of the framework. Additionally, the analysis of the experiment results has led to several non-trivial insights on concept drift detection techniques. Firstly, the evaluation results have shown that increasing levels of parallelism in the model have mixed effects on the evaluated techniques. Some techniques become more accurate and others become less accurate. Secondly, the type of change in the process (e.g., add or remove a fragment) has a much larger effect on the accuracy of the approaches than the actual type of drift (i.e., sudden or gradual). Thirdly, the type of drift transition function (i.e., linear or exponential) has no effect on the accuracy of the approaches considered in the experiments.

Lastly, the *VariantFinder* technique introduced in Chapter 6 was successfully used to detect concept drift, even though it was not its initial usage purpose. The *VariantFinder* technique had a better accuracy yet a worse efficiency (i.e., computation time) than all the other evaluated techniques in the context of the experiments performed in this chapter.

Overall, all the techniques considered in the experiment present a trade-off between accuracy and computation time, and users should carefully choose which technique to use depending on the specific analysis settings.

A limitation of this experiment is that is merely theorical, as it covers all types of concept drift equally. An empirical study would be interesting e.g., measuring which types of concept drit are more common in real life, and then analyze how the different techniques can deal with them.

Part IV

Case Studies

Chapter 9

SLA Compliance Analysis in a Claim Management Process

Disclaimer: The data used in this chapter is subject to a non-disclosure agreement signed by the author and by *Telefónica*. Therefore, the data is not publicly available.

Service-level agreements (SLA) are often signed by organizations in order to maintain a certain level of service by the use of penalties or rewards. Service-levels below the agreement usually lead to the provider of the service being penalized. A tasty example of an SLA are those pizza delivery places that will deliver your pizza in less than 30 minutes, or it's free.

SLA compliance can be related to the performance of the service's underlying processes. Processes that perform better will lead to an increased SLA compliance. Therefore, it is important to analyze the different factors that affect SLA compliance and relate these factors to variants of the process in order to learn the best practices and identify their inefficiencies.

By using the tools and techniques proposed in Part II, we can discover, compare and analyze the process variants that are related to lower and higher SLA compliance. This chapter presents an example of this analysis in the context of a real use-case using real process data as input.

The remainder or this chapter is structured as follows. Section 9.1 introduces the context of this case study. Section 9.2 describes the experiments performed, and the obtained results. Section 9.3 discusses the results and their

impact on the company. Finally, section 9.4 concludes the chapter.

9.1 Context

Telefónica is a Spanish broadband and telecommunication provider with its head quarter located in Madrid. It operates in Europe, Asia, North and South America, being one of the biggest telecommunication providers in the world. In the remainder of this chapter, we will refer to Telefónica as simply "the company". The company offers different solutions for commercial as well as business customers. For their corporate services, the company runs a system that manages *claims* (usually created by the customer) from different services in the form of tickets.

The remainder of this section describes the claim management process, the dataset used, the process SLAs defined by the company, and the purpose of the analysis in the form of process questions.

9.1.1 Process Description

The *claim management process* relates to the actions executed by the company in order to handle claims of customers that have a problem with a given service. In the process of solving the claim, the ticket can be in different states. Figure 9.1 shows a hand-made model of the claim management process, which was made by the company.

The process is described as follows: Claims are created usually by the customer (i.e., the "New" state is reached), then they are activated when an employee starts working on the claim (i.e., the "Active" state). When there is an interruption of the service, the service needs to be restored first (i.e., the "Restored" state) and the employees will work on the problem that caused it until it is solved (i.e., the "Solved" state) and finally the claim is closed (i.e., the "Closed" state). Claims can be cancelled (i.e., the "Cancelled" state) or delayed (i.e., the "Delayed" state) anytime, regardless of the current state. Delayed claims can be reactivated at any time as well. Delays are commonly caused by missing information that the customer has to provide in order to identify the problem.

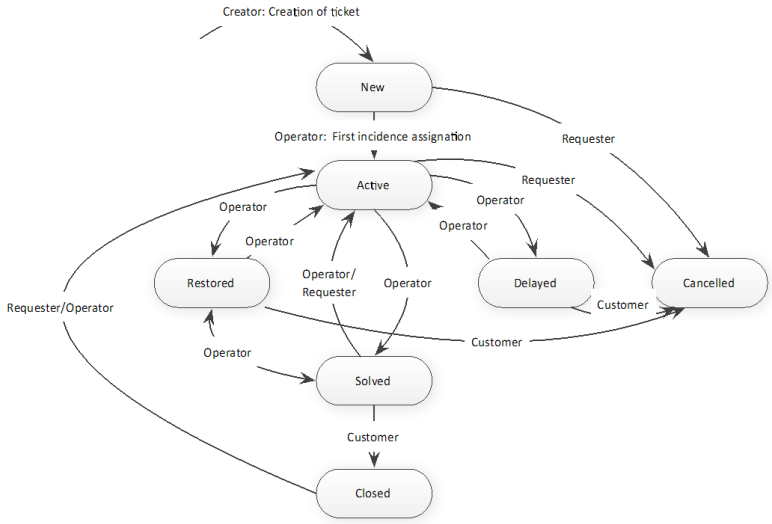


Figure 9.1: Hand-made model of the flow of a claim. The boxes represent the possible states of a claim. The arrows indicate the possible state changes. This model was provided by the company.

9.1.2 Event Data

The data presented in this chapter (provided by the company) contains the logged execution of the claim management process in the form of CSV (i.e., comma-separated-values) files. It contains 8296 claims (i.e., cases) and 40965 state changes (i.e., events) in total, with an average of 5 state changes per claim. This data was recorded between January 2015 and December 2016 for three services, codenamed: GSIM, JASPER, and SM2M. Table 9.1 illustrates the event attributes contained in the data.

The data includes four different possible severities for a claim, based on the impact and urgency of the issue: critical, major, minor and slight.

Note that the attribute *Time_of_week* is used to bin timestamps (i.e., the *Date* attribute) into six values that indicate different times of the week: LV_M (weekday mornings), LV_T (weekday evenings), LV_N (weekday nights), SD_M (weekend mornings), SD_T (weekend evenings), and SD_N (weekend nights).

Also note that claims can be created by the customer (i.e., “isChild” = 0) or

Table 9.1: Event attributes contained in the data

Name	Description
Id	the id of a claim
Service	the service related to the claim
Date	the timestamp of activity
Modified_by	the person that executed the activity
Responsible.Operator	the person responsible for the claim's SLAs
Responsible.group	the group responsible for the claim's SLAs
Assigned.group	the group to which the claim was assigned
Operation	the activity that was executed
initial.Severity	the initial severity of the claim, defined when the claim was created
update.Severity	the current severity of the claim[1]
Severity	the final severity of the claim, when the claim was solved
Originator	the person that created the claim
Originator.Group	the group to which the originator belongs
Time_of_week	the time of the week, based on the Date attribute
IsChild	whether the claim is a subdivision of another claim (1) or not (0)

can be created by an employee as a subdivision of a bigger *parent* claim (i.e., "isChild" = 1). The company's domain experts indicated that the *child* claims are given priority because their processing time affects the SLA's of the parent claim.

It is important to state that the raw data delivered to us did not include any explicit information about SLA compliance.

9.1.3 SLAs

The company defined several SLAs based on process performance metrics, on the service and on the severity of the claim, which define how quickly the company has to manage claims in order to not being penalized. The process performance metrics considered are listed as follows.

Response time : The time it takes for an employee to start working on the

ticket. This is measured as the time passed since a claim was created in the system (i.e., reached the "New" state) until it has been activated (i.e., reached the "Active" state for the first time).

Restoration time : The time it takes until the service is working again in the case of service interruption. This is measured as the time passed since a claim was created in the system (i.e., reached the "New" state) until the service is restored (i.e., reached the "Restored" state). Alternatively, if there was no interruption of the service, then the time between the "New" and "Solved" states is considered instead.

Resolution time : The time it takes until the problem that caused the malfunction or interruption of the service is solved. This is measured as the time passed since a claim was created in the system (i.e., reached the "New" state) until the problem is solved by the company (i.e., reached the "Solved" state).

Each of these performance metrics define an SLA for each of the three services and four severities of a claim, as illustrated in Table 9.2.

It is important to note that when a claim is in the "Delayed" state, the time spent there does not affect SLA compliance. Claims are normally delayed so that the SLAs defined for the services are not affected by situations out of the company's control.

The company defined that the *restoration time* SLAs for minor and slight claims related to the SM2M and GSIM services are measured only during agreed standard business hours (from 9 am to 5 pm, Spain local time). Also Note that the company defines *resolution time* SLAs only for critical and major severity claims related to the SM2M service.

9.1.4 Analysis Purpose

The company asked us to analyze the compliance of the process with respect to the SLAs defined above in order to obtain insights about which are the best practices and which are the points of improvement of the process, with the purpose of improving SLA compliance.

Concretely, the company asked the following questions about the process:

PQ1: What is the overall SLA compliance of the process?

PQ2: Which claims are less (or more) likely to comply with their SLA?

Table 9.2: SLAs defined for the claim management process.

(a) Response Time SLAs

Service	Severity			
	Critical	Major	Minor	Slight
GSIM	15 min	45 min	120 min	180 min
Jasper	15 min	45 min	120 min	180 min
SM2M	15 min	30 min	60 min	60 min

(b) Restoration Time SLAs

Service	Severity			
	Critical	Major	Minor	Slight
GSIM	6 hrs	24 hrs	48 hrs	80 hrs
Jasper	6 hrs	24 hrs	-	-
SM2M	5 hrs	12 hrs	48 hrs	80 hrs

(c) Resolution Time SLAs

Service	Severity			
	Critical	Major	Minor	Slight
GSIM	-	-	-	-
Jasper	-	-	-	-
SM2M	96 hrs	480 hrs	-	-

PQ3: What are the differences between claims that complied with their SLA and claims that did not?

In the next section we will address each one of these questions in detail.

9.2 Experiments

This section reports on the experiments performed in order to answer each of the three process questions asked by the company. As stated before in Section 9.1.2, the raw data does not include explicit SLA compliance information. Therefore, we first need to perform a data preparation step. The purpose of this step is to add explicit SLA compliance information to the data, which will be

used in the rest of the experiments.

9.2.1 Data Preparation

As mentioned earlier, when a claim is in the "Delayed" state, the time spent there does not affect SLA compliance. Moreover, some SLAs are measured only during agreed standard business hours (from 9 am to 5 pm, Spain local time). However, the recorded timestamps (i.e., the *Date* attribute) do not account for this and just record the timestamp when the state of a claim changed. Hence, we cannot directly use the timestamps to calculate SLA compliance.

SLA-aware Time Measurement

To address the issue described above, we created a new event attribute named *SLA time*, which is based on the timestamp attribute with some modifications: Initially, the *SLA time* attribute has the same value as the timestamp attribute for all the events related to a claim. Every time that the claim reaches the "Delayed" state, the value of the *SLA time* attribute is copied to subsequent events until the claim leaves the "Delayed" state. Then, for all following events, the value of the *SLA time* attribute is calculated as the difference between the timestamp of the event and the time spent by the claim in the "Delayed" state. If a claim is delayed again, then the same procedure is applied again.

Additionally, we modified the *SLA time* attribute further in the case of claims whose SLAs are measured only during agreed standard business hours (i.e., weekdays from 9 am to 5 pm, Spain local time). In such cases, the *SLA time* attribute is modified to consider only the time spent within agreed standard business hours. Note that local public holidays do not affect standard business hours since people processing the claims are located in different parts of the world.

As a result, the newly-created *SLA time* attribute can be used for the purpose of SLA compliance checking.

Checking SLA Compliance

As mentioned earlier, SLAs are defined for each service and severity of claims, as described in Table 9.2. For example, a major severity claim related to the GSIM service has a *response time* SLA of 45 minutes, a *restoration time* SLA of 24 hours, but has no *resolution time* SLA.

In this case study, SLA compliance is measured on the earliest possible occasion. This means that if a claim reaches a state related to a SLA several times, only the first one is considered. For example, the *response time* SLA compliance is checked only when a claim reaches the "Active" state for the first time. The same applies for the *restoration time* and *resolution time* SLAs, related to the "Restored" and "Solved" states.

Note that, in some cases, SLA compliance cannot be checked because the claim never reaches the state related to such SLA. For example, a claim that is cancelled after being created and does not reach the "Active" state, or a claim being processed that has not reached such state yet. Therefore, the *response time* SLA cannot be checked for such claim.

We used the *SLA time* attribute created before to check, for each claim and for each SLA defined for it, if it complied with the SLA (1) or not (0). Concretely, for each claim we created three new case-level attributes: response compliance, restoration compliance and resolution compliance. These attributes indicate if the claim complied or not with the *response time*, *restoration time* and *resolution time* SLAs defined for it, respectively.

With this SLA compliance information added explicitly to the dataset, we can proceed to answer the process questions described before.

9.2.2 Overall SLA Compliance Diagnostic

This section addresses the first process question: *What is the overall SLA compliance of the process?* (PQ1).

We do this analysis by looking at the overall compliance of the response time, restoration time and resolution time SLAs through the use of simple descriptive statistics. Note that this question can be answered without using process-aware techniques, because now the SLA compliance information is explicitly mentioned in the following data attributes: *response compliance*, *restoration compliance* and *resolution compliance*.

Firstly, we analyzed if there is a "cascade effect" between the different SLAs. Intuitively, claims that complied with their *response time* SLA should be more likely to comply with their *restoration time* SLA than claims that did not comply with their first SLA. However, the data shows that there is no clear correlation between these SLAs. On the one hand, from the 182 claims that failed their *response time* SLA, 172 claims (94% approx) complied with their *restoration time* SLA. On the other hand, from the 6723 claims that complied with their *response time* SLA, 6088 claims (91% approx) complied with their *restoration*

time SLA. The statistical tests indicated that the difference between the *response time* SLAs of both groups is not significant[2].

Since the compliance of the SLAs defined for the process are not correlated (e.g., a claim can fail its response time SLA but comply with its restoration time SLA) we proceed to analyze them separately.

With respect to the *response time* SLA, from the 8296 claims recorded in the dataset, 91 claims never reached the "Active" state, thus are not included in the analysis. From the 8205 claims that did reach the "Active" state, only 7828 claims (95% approx) complied with their *response time* SLA.

We are also interested in analyzing if there are SLA compliance differences for the three services and four claim severities found in the data. Figure 9.2 shows the overall compliance with the Response SLAs for different services and claim severities. We can observe that the SM2M service has a much higher

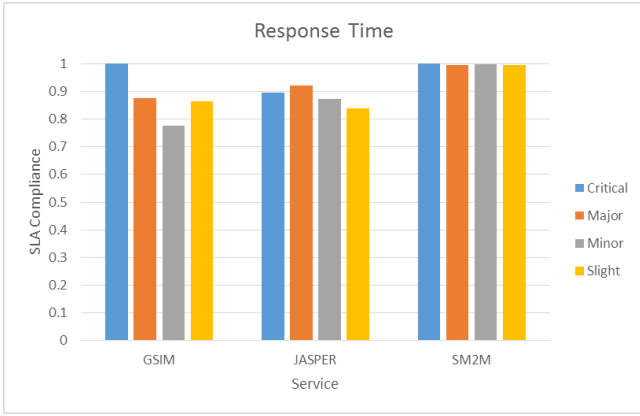


Figure 9.2: Response Time SLA compliance (avg) by service and severity of claims.

response time SLA compliance than the other two services.

With respect to the *restoration time* SLA, from the 8296 claims recorded in the dataset, 1391 claims never reached the "Restored" nor the "Solved" state, thus are not included in the analysis. From the 6905 claims that reached the "Restored" state (or the "Solved" state if there was no service interruption), 6342 claims (91% approx) complied with their *restoration time* SLA.

Like the previous SLA, we also analyzed if there are SLA compliance differences for the three services and four claim severities found in the data. Fig-

[2]The statistical test used was the rank-based non-parametric Mann-Whitney U test.

ure 9.3 shows the overall compliance with the Restoration SLAs for different services and claim severities. We can observe that in the case of the GSIM and

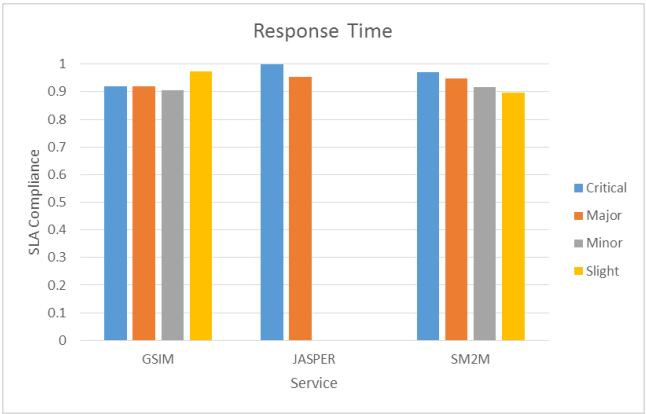


Figure 9.3: Restoration SLA by service and severity of claims.

JASPER service, the *restoration time* SLA compliance is better (in average) than the *response time* SLA. We can also observe that all services and severities have a *restoration time* SLA compliance of approximately 90% or more.

Finally, we analyzed the *resolution time* SLA. Note that this SLA is only defined for the SM2M service and only for critical and major severity claims. From the 8296 claims recorded in the dataset, 212 claims are related to the SM2M service and have a critical or major severity, and only 178 of such claims reached the "Solved" state. From these, 128 claims (72% approx) complied with their *resolution time* SLA.

Unlike previous SLAs, the *resolution time* SLA is defined only for one service and two severities. Because of this reason, and because the number of claims that were checked for compliance with the resolution time SLA is low (thus, not very representative), we did not split them further in this analysis.

9.2.3 Correlating Claims to SLA Compliance

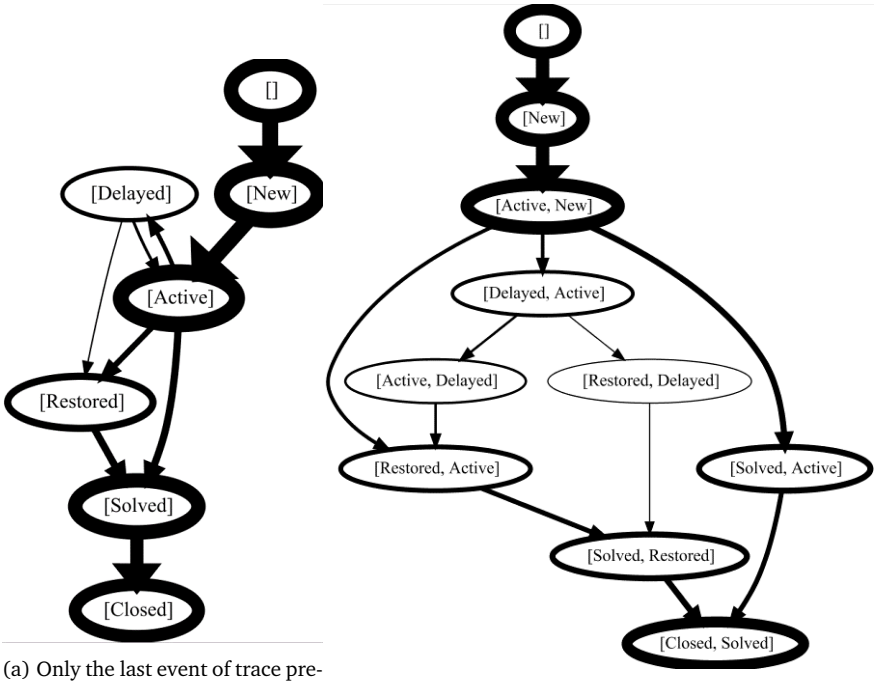
This section addresses the second process question: *Which claims are less (or more) likely to comply with their SLA?* (PQ2).

In order to answer this question, we need to find the variants of the process that are more correlated to SLA compliance. For this purpose, we use the pro-

cess variant detection technique introduced in Chapter 6. This technique allow us to find process variants with statistically-significant differences in the data.

Before finding the process variants, we need to choose the right abstractions to represent the claim management process.

First, we created a transition system that represents the process based on an abstraction where the last event of trace prefixes is considered (i.e., only directly-follows relations are considered). We used the following state and transition abstraction: given the event log L, a trace $\sigma \in P_L$ and an event $e \in E_L$, $r^s(\sigma) = att_n(\sigma(|\sigma|))$ and $r^a(e) = att_n(e)$. This is illustrated in Figure 9.4a.



(a) Only the last event of trace prefixes is considered.

(b) The last two events of trace prefixes are considered.

Figure 9.4: Transition systems that represent the behavior of the claim management process using different abstractions.

We can observe a natural loop between the “Delayed” and “Active” states. This is explained by the fact that claims can be delayed several times. However, by using this abstraction we cannot distinguish between claims that are

solved or restored with delays and the claims that are solved or restored without delays. Therefore, we created a new transition system that represents the process based on the abstraction where the last two events are considered: $r^s(\sigma) = \langle att_n(\sigma(|\sigma|)),\ att_n(\sigma(|\sigma|-1))\rangle$ and $r^a(e) = att_n(e)$. Unlike the previous transition system, we used this abstraction in this experiment to include more than just directly-follows relations. This transition system is illustrated in Figure 9.4b.

We can observe that now there is a clear distinction between claims that have been delayed and those that have not been delayed (i.e., non-delayed claims go from the "Active, New" state directly to the "Restored, Active" or "Solved, Active" states).

Starting from this transition system, we looked for process variants in the process. The remainder of this section describes the analysis performed for each SLA.

Finding Variants over Response Time SLA Compliance

In this experiment, we aim to find process variants that are correlated to the *response time* SLA using the Variant Finder tool presented in Chapter 6. For this purpose, the response compliance attribute is selected as the dependent variable. All the other attributes are selected as independent variables. We only analyzed the variants found in the "New" and "Active, New" states presented in Figure 9.4b, because the other states are not related to the response time SLA.

The process variants found in the "New" state are shown in Figure 9.5. We

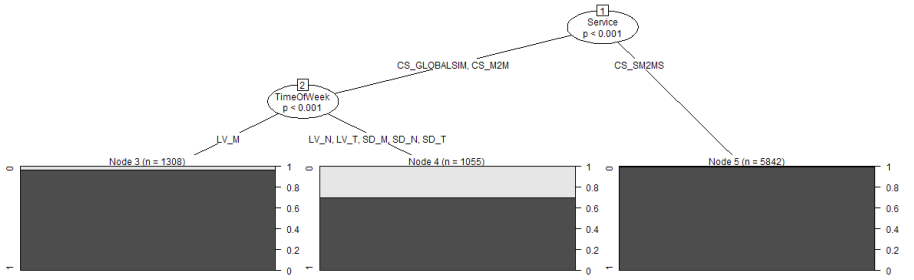


Figure 9.5: Results of the Process Variant Finder tool: attributes correlated to the *response time* SLA in the "New" state.

can observe that the attribute that is most correlated to the response compliance

in the "New" state is the Service attribute. The results indicate a clear response compliance difference between the GSIM and JASPER services, and the SM2M service, with the latter having better response compliance. We can also observe that within the GSIM and JASPER services, there is a clear difference between claims that were created on weekday mornings (i.e., LV_M) and claims created in any other time of the week: claims created in weekday mornings have a much higher response compliance.

Now, we proceed to analyze the response compliance variants found in the "Active, New" state. The process variants found in the "Active, New" state are shown in Figure 9.6.

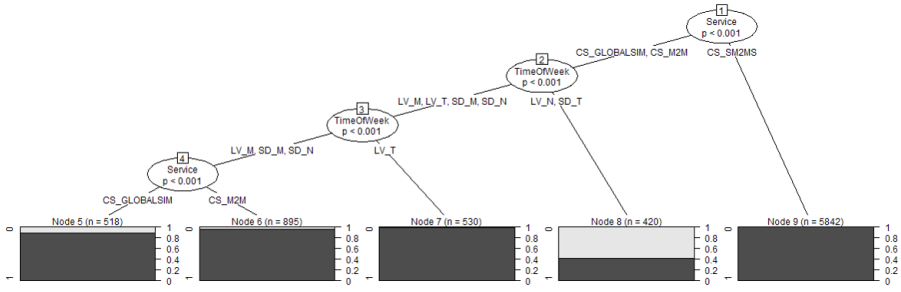


Figure 9.6: Results of the Process Variant Finder tool: attributes correlated to the *response time* SLA in the "Active, New" state.

Similarly to the previous analysis, we can observe that in the "Active, New" state there is also a considerable response compliance difference between the GSIM and JASPER services, and the SM2M service. However, in this case we can observe that within the GSIM and JASPER services, there is a clear difference between claims that were processed on weekday nights (i.e., LV_N) or in weekend evenings (i.e., SD_T), and claims that were processed on other times of the week, with the former having a lower response compliance than the latter.

Surprisingly, at this point of the process the severity of the claim does not seem to have a strong correlation with the response compliance.

Finding Variants over Restoration Time SLA Compliance

In this experiment, we aim to find process variants that are correlated to the *restoration time* SLA using the Variant Finder tool presented in Chapter 6. This

time, the dependent variable is set as the restoration compliance attribute. All the other attributes are selected as independent variables.

We found several variants in the "Delayed, Active" state shown in Figure 9.4b. These are illustrated in Figure 9.7.

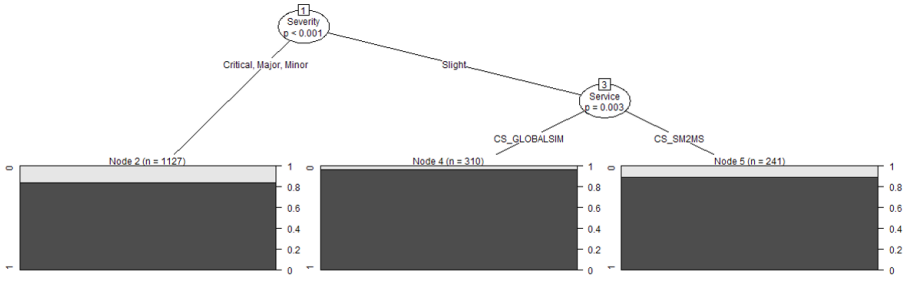


Figure 9.7: Results of the Process Variant Finder tool: attributes correlated to the *restoration time* SLA in the "Delayed, Active" state.

For all the claims that get delayed at least once (i.e., the claims that reach the "Delayed, Active" state, which is approximately 20% of all claims) we can observe that claims with a slight severity have a better restoration compliance than other more-severe claims. Moreover, from the claims with a slight severity, those that are related to the GSIM service have a better restoration compliance than claims related to the SM2M service.

Additionally, we found process variants in the transition between the "Active, New" and the "Solved, Active" states (see Figure 9.4b). These are illustrated in Figure 9.8. These variants are related to claims that were solved without being delayed and without a service interruption (i.e., no restoration was needed). These claims ammount to approximately 50% of all claims.

We can observe that claims that are derived from other bigger claims (i.e., isChild = 1) have a lower restoration compliance than claims that are not derived from other claims.

Finding Variants over Resolution Time SLA Compliance

In this experiment, we aim to find process variants that are correlated to the *resolution time* SLA using the Variant Finder tool presented in Chapter 6. This time, the resolution compliance attribute is selected as the dependent variable. All the other attributes are selected as independent variables. Note that this

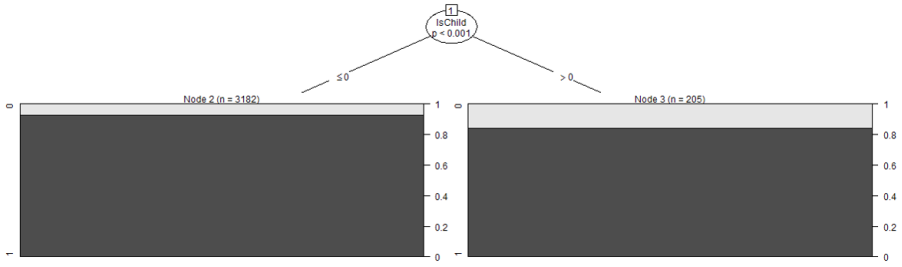


Figure 9.8: Results of the Process Variant Finder tool: attributes correlated to the *restoration time* SLA in the transition between the "Active, New" and the "Solved, Active" states.

SLA is only defined for the SM2M service and only for critical and major severity claims. In total, only 178 claims have a value of 0 or 1 for the resolution compliance attribute.

Figure 9.9 shows the process variants found in the "Closed, Solved" state, which considers all claims that were solved and then closed, regardless of delays and service restorations. These correspond to approximately 90% of the claims that have a value for the resolution compliance attribute.

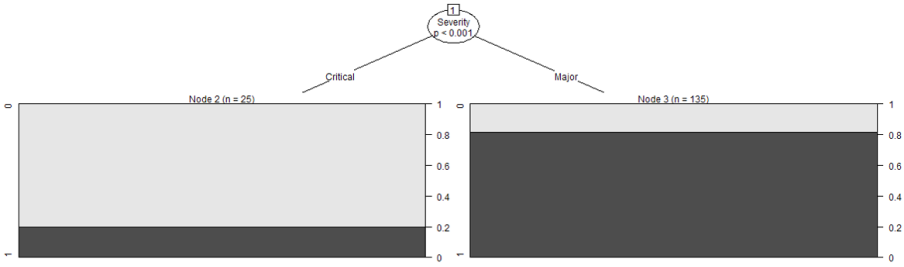


Figure 9.9: Results of the Process Variant Finder tool: attributes correlated to the *resolution time* SLA in the "Closed, Solved" state.

We can observe that the severity attribute has the strongest correlation with the resolution compliance attribute. Claims that have a critical severity have a lower resolution compliance than claims with a major severity.

Now that we know which claims are less (or more) likely to comply with

their SLA, we proceed to find the most significant differences between them.

9.2.4 Comparing SLA-Compliant and SLA-Non-Compliant Claims

This section addresses the third process question: *What are the differences between claims that complied with their SLA and claims that did not?* (PQ3).

In the previous section we found several process variants that are closely related to data attributes. An interesting analysis is to compare these variants against each other in order to find the differences and similarities between them. We did compare the different services and claim severities and found several differences. This analysis was presented to the company and positive feedback was obtained. However, such analysis is out of the scope of the questions asked by the company. So, in this section we will focus on the actual question mentioned above.

As mentioned in the data preparation step (see Section 9.2.1) three attributes were created in order to explicitly denote wether claims complied with their SLAs or not: *response compliance*, *restoration compliance* and *resolution compliance*. These attributes are related to the response time, restoration time and resolution time SLAs respectively. For any claim, the *response compliance*, *restoration compliance* and *resolution compliance* attributes can have a value of 1 or 0, depending on if it complied with the corresponding SLA or not.

In the remainder of this section, we will compare claims based on their compliance to the different SLAs defined for the process.

Comparing Variants over Response Time SLA Compliance

In order to compare claims that comply with their response time SLA or not, we first split the data based on the *response compliance* attribute using the process cube tool introduced in Chapter 3. As a result, we obtained two process variants: one containing all the claims (7828) that complied with their response time SLA (i.e., response compliance = 1) and another containing all the claims (377) that failed to comply with such SLA (i.e., response compliance = 0).

We compared these two process variants against each other using the process comparator technique presented in Chapter 5. Note that for building the underlying transition system, we used the same abstraction used in the previous section (i.e., using the last two events of trace prefixes).

It is important to note that the response time SLA measures the very first part of the process (i.e., from the “New” state to the “Active, New” state). At this point of the process, nothing else has happened in the claim yet, but when

comparing the two process variants, we found several control-flow differences between them, as illustrated in Figure 9.10.

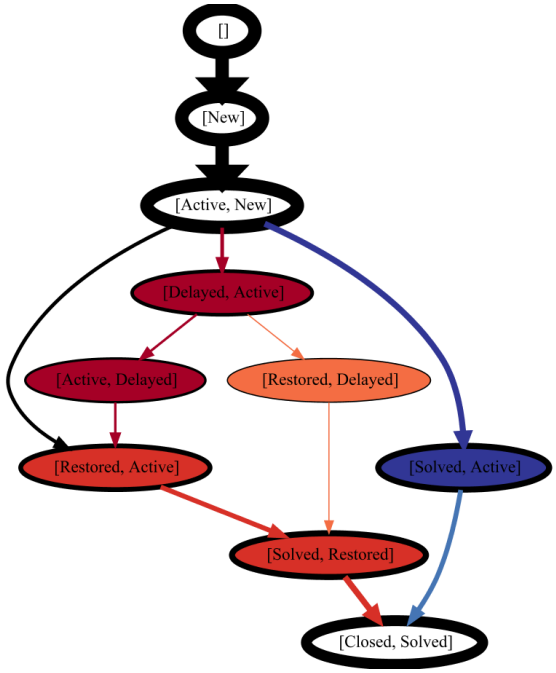


Figure 9.10: Control-flow comparison results between claims that complied with their *resolution time* SLA and claims that did not.

Note that, in this figure, blue colors represent states or transitions in which the compliant claims (i.e., response compliance = 1) reach such state or transition with a frequency that is significantly higher than non-compliant claims (i.e., response compliance = 0). Red colors represent the opposite.

We can observe that from the "Active, New" state (where the response time SLA is measured) we have mainly three next possible states: "Solved, Active", "Restored, Active" and "Delayed, Active".

For claims that were directly solved and there was no interruption of the service (i.e., reached the "Solved, Active" state), we can observe that 43% of the compliant claims followed this path and got directly solved without delays. On the other hand, only 1% of the non-compliant claims followed this path.

The difference is obviously significant, hence the blue color of the transition between the "Active, New" and the "Solved, Active" states.

For claims that were restored (i.e., reached the "Restored, Active" state), we noticed that 26% of the compliant traces and 20% of the non-compliant traces go through this path. However, this difference is not statistically significant, hence the black color of the transition between the "Active, New" and the "Restored, Active" states.

For the claims that were delayed (i.e., reached the "Delayed, Active" state), we observe that while 26% of the compliant claims followed this path, 75% of the non-compliant traces followed this path as well. The difference is again obviously significant, hence the red color of the transition between the "Active, New" and the "Delayed, Active" states.

These differences tell us that non-compliant claims are more prone to delays, whereas compliant claims are more prone to get solved without delays or service interruptions.

From the data available for this analysis it is not easy to guess what the reason behind this difference is. Perhaps the information provided in the claim is related to them getting delayed. For example employees could choose to start working on easier claims over other claims with difficult problems or insufficient information.

Comparing Variants over Restoration Time SLA Compliance

Similarly to the previous analysis, we split the data based on the *restoration compliance* attribute (which measures compliance with the restoration time SLA) using the process cube tool introduced in Chapter 3. As a result, we obtained two process variants: one containing all the claims (6342) that complied with their restoration time SLA (i.e., restoration compliance = 1) and another containing all the claims (563) that failed to comply with such SLA (i.e., restoration compliance = 0).

Again, we compared these two process variants against each other using the process comparator technique presented in Chapter 5.

Figure 9.11 shows the results of the control-flow comparison between these variants.

We can first observe that there are less significant differences between these variants than in the previous analysis. However, the "Delayed, Active" state presents an interesting analysis opportunity. We can observe that 19% of the compliant claims and 27% of the non-compliant claims reach this state. The difference is significant, hence the red color of the state. Then, we compared

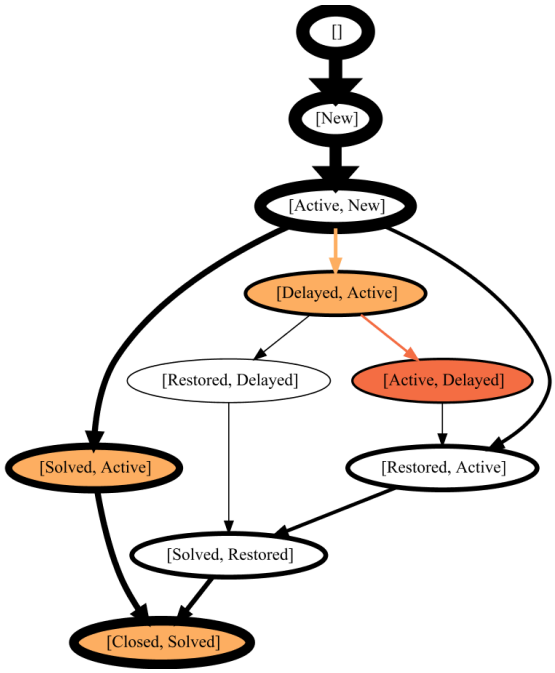


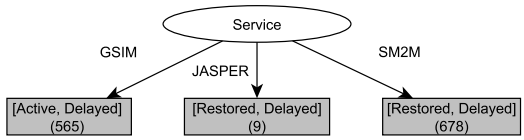
Figure 9.11: Control-flow comparison results between claims that complied with their *restoration time* SLA and claims that did not.

the business rules (represented as decision trees) that can be used to predict the next state to be reached in both variants (see Section 5.2.2 for more info on business rule comparison).

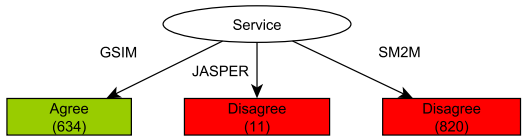
Figure 9.12 shows the results of the business rules comparison between the two variants in the state "Delayed, Active".

Note that the decision tree learned form the non-compliant claims is not shown in Figure 9.12 because it classifies all claims in the "Active, Delayed" state.

We can observe that after the "Delayed, Active" state, most compliant claims related to the GSIM service get activated again (i.e., they reach the "Active, Delayed" state). This is not the case for the JASPER and SM2M services, where compliant claims get restored directly after the delay (i.e., they reach the "Restored, Delayed" state). This means that claims are being restored while being



(a) Decision tree learned from the compliant claims.



(b) Agreement/disagreement decision tree.

Figure 9.12: Decision trees learned in the decision point (state) "Delayed, Active".

delayed.

Since all non-compliant claims tend to get activated (i.e., reach the "Active, Delay" state), the agreement/disagreement tree is quite obvious: only for claims related to the GSIM service there is an agreement, hence they follow the same control-flow (i.e., the "Active, Delayed" state). Claims related to the other two services reach different states depending if they are compliant (i.e., the "Restored, Delayed" state) or not (i.e., the "Active, Delayed" state).

In theory, claims that are in a delayed state cannot reach the restored state directly: they have to go through the active state. As we have shown in the above analysis, this is not the case. This phenomenon will be discussed in detail in Section 9.3

Comparing Variants over Resolution Time SLA Compliance

This time we split the data based on the *resolution compliance* attribute (which measures compliance with the resolution time SLA) using the process cube tool introduced in Chapter 3. As a result, we obtained two process variants: one containing all the claims (128) that complied with their resolution time SLA (i.e., resolution compliance = 1) and another containing all the claims (50) that failed to comply with such SLA (i.e., resolution compliance = 0). Note that the resolution time SLA is only defined for the SM2M service and only for critical and major severity claims.

Again, we compared these two process variants against each other using the process comparator technique presented in Chapter 5.

First we compared the variants from the control-flow perspective, but no significant differences were found. Then, we compared the variants from the performance perspective. Figure 9.13 shows the results of the performance comparison between compliant and non-compliant claims based on their elapsed time.

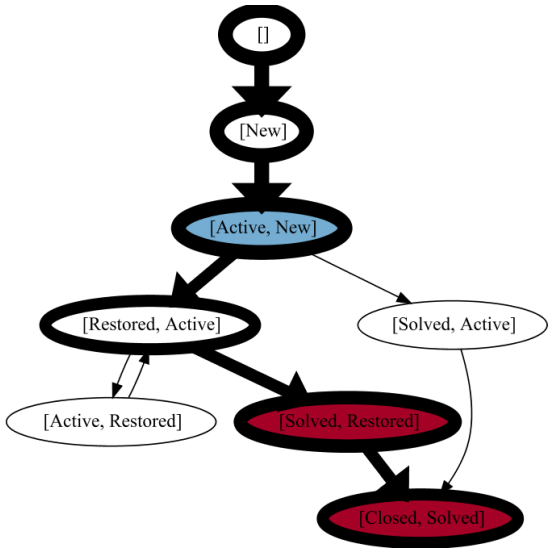


Figure 9.13: Performance comparison results between claims that complied with their *resolution time* SLA and claims that did not.

We can observe that there are three states that contain statistically significant differences in terms of elapsed time: “Closed, Solved”, “Active, New’ and “Solved, Restored”.

First of all, the “Closed, Solved” state (highlighted in red) presents significant differences, but such differences are not interesting for the purpose of our analysis because the closure of claims is not considered by any SLA. This is because the closure of claims is done by the customer, not the company.

The “Active, New” state (highlighted in blue) is reached by compliant claims in 2.5 minutes in average, while non-compliant claims reach this state much faster: 46 seconds in average. Even though the difference is significant, both

variants still have a very low response time, which is even lower than all the defined SLAs for the process (see Table 9.2).

With respect to the "Solved, Restored" state, besides stating the obvious (i.e., non-compliant claims take longer to get solved) we have to look at the previous state: "Restored, Active". Notice that such state is not colored, hence there are no significant differences in terms of elapsed time between compliant and non-compliant claims. This means that the compliance or non-compliance of the resolution time SLA is mostly determined after the restoration of the service. In fact, the time elapsed between the "Restored, Active" and the "Solved, Restored" states is approximately 6 days for claims that ended up complying with the resolution SLA and 28 days for non-compliant claims. Given the available data, it is difficult to know the reason for such performance differences; It can be related to the information contained in the claim (e.g., the type of problem).

9.3 Discussion: The Delayed State

As mentioned before, we observed that there is a difference on how the delayed state is used in different claims. Recall that time in this state is stopped and, therefore, it does not count towards any SLA. As shown in Figure 9.1, it is clear that delayed claims can either become active again or can get cancelled by the customer. The first scenario means that a ticket was delayed because it required some input from the customer. After the input is obtained, the claim is set to active again. Then, work on the claim can continue. This means that the actual work is done in the active state, which naturally counts for SLA compliance.

In our analysis, however, we noticed that an irregularity occurs: claims are being restored directly after being delayed (i.e., without being activated). In this scenario, a claim is delayed because of the same reasons. Nevertheless, it is seems that the actual work is done while the claim is in the delayed state, and not in an active state, hence the time does not count towards the SLA. Employees use this trick to pause the time and work on cases. When things are working again, they switch to the restored state. Of course, this is not in line with the purpose of the delayed state, which is to protect the company from being penalized for insufficient information in a claim.

This behaviour can be observed in cases across all services and severities, and it is clearly beneficial for the company in the sense that such "time savings" positively impact SLA compliance. Therefore, less cases fail their SLAs and the company has to pay less compensation.

We presented these obtained results and analysis to the company. They

were surprised that this situation existed, and rapidly fixed the problem. As a consequence, a change was made in the process: in May 2017, the system that supports the process was modified so that claims were no longer allowed to go directly from being in a delayed state to a restored state. From that point in time, all delayed claims have to become active before a service restoration can be done. In the future, we want to analyze the actual impact of such change in the process on SLA compliance.

9.4 Conclusion

In this chapter, we presented an application of the tools and techniques introduced in this thesis to analyze SLA compliance of a claim management process using real process data. Concretely, we found insights that are not obtainable using other current tools and techniques. We used process cubes (see Chapter 3) to split the data as needed for the different experiments. We used the process variant finder (see Chapter 6) to find the attributes contained in the data that are correlated to SLA compliance. Finally, we used the process comparator (see Chapter 5) to compare the claims that complied with their SLAs and those that didn't.

We analyzed the compliance of the process for the different SLAs defined by the company, obtaining multiple useful insights that were communicated to the company. For example, we discovered that claims related to the SM2M service have a good overall response time SLA compliance. However, for the GSIM and JASPER services, we discovered that the time of the week (e.g., weekday/weekend, morning/evening) is closely related to the response time SLA compliance: claims created in weekday mornings had much higher compliance with this SLA, regardless of the severity of the claim.

Furthermore, we discovered a misuse of the delayed state, in which employees can work on a delayed claim without counting the time for SLA compliance. The company immediately corrected this after we informed them about this phenomenon, by making changes in the system that supports the process so that this behavior is not allowed anymore.

Chapter 10

Business Process Reporting in Education

Disclaimer: The data used in this chapter is subject to a non-disclosure agreement signed by the author and by *Eindhoven University of Technology*. Therefore, the data is not publicly available.

Business Process Reporting (BPR) refers to the provision of structured information about processes in a regular basis, and its purpose is to support decision makers. Reports can be used to analyze and compare processes from many perspectives (e.g., behavior, performance, costs, time). In order to be effective, BPR presents some challenges:

1. It should provide insights using metric-based characteristics (e.g., bottlenecks, throughput time, resource utilization) and behavioral characteristics (e.g., deviations, frequent patterns) of processes.

2. It should be repeatable (i.e., not require great efforts to repeat the analysis).

3. It should be able to analyze the data with different granularity levels (i.e., analyze an organization as a whole or analyze its branches individually).

This chapter shows through a case study how *process mining workflows* (see Chapter 4) and *process cubes* (see Chapter 3) can be concretely used for business process reporting, addressing the three challenges mentioned above.

The case study presented in this chapter refers to a business-process reporting service at Eindhoven University of Technology. The service produces a report each quartile (i.e., two-month academic period) for each course that is provided with video lectures. The report is sent to the responsible lecturer and provides insights about the relations between the students' usage of video lectures and their final grades on the course, among other educational data analysis results.

The usefulness of the reports is evaluated with dozens of lecturers throughout two evaluation rounds in different academic periods. During the first evaluation round, an initial set of reports was sent to lecturers and feedback was collected through an evaluation form. The feedback was used to restructure the report. Then, a set of restructured reports was sent to lecturers and a group of them were interviewed to assess if the insights contained in the report were better perceived. The results show that, indeed, this is the case.

The remainder of this chapter is organized as follows. Section 10.1 provides an overview of the case study and discusses related work. Section 10.2 discusses the structure of the reports sent and the results of the two evaluation rounds with the lecturers. Finally, Section 10.3 concludes the chapter.

10.1 Context

The *Eindhoven University of Technology* is a top-ranked Dutch university. Despite being relatively young (it was founded in 1956) it is ranked as one of the top 100 universities in the world[1]. It has over 11.000 students from more than 80 nationalities.

Amongst many services, Eindhoven University of Technology provides video lectures for many courses for study purposes and to support students who are unable to attend face-to-face lectures for various reasons. There are three different types of video lectures. The first are recordings of real-life lectures of the actual (or previous) academic period. The second type replace the actual real-life lectures in courses that implement flipped-classroom learning styles. Finally, the third type are complementary to the real-life lectures of the course, as is the case of e.g., video tutorials. Student usage of video lectures and their course grades are logged by the University's IT systems.

The remainder of this section describes the video lecture usage process, the dataset used, the purpose of this case study and related works.

[1] 99th place according to QS Ranking 2018

10.1.1 Process Description

For any given course that offers video lectures to the students that take it, the video lecture usage process considers all the views of the video lectures of a given course, where each student that watches video lectures is related to a different case of the process. Hence, each video lecture view is considered as an activity of the process. Video lectures can be watched by students in any order and as many times as they want. Finally, the students take a final exam for the course, which in this case is the final activity of the video lecture usage process.

The video lectures are numbered by the lecturer in the sense that they define the ideal order in which students should watch them, e.g., from lecture 1 to lecture n. However, within the ideal order, students can watch the same video lecture several times in a row. This could be because of many partial views that, in total, complete the video lecture, or because students watch it multiple times due to the complexity or difficulty of its contents. Figure 10.1 describes the ideal video lecture usage process followed by students for a given course with n video lectures.

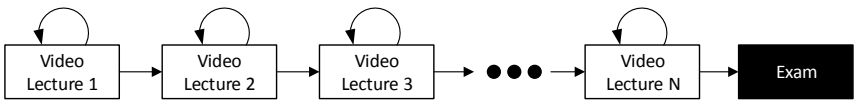


Figure 10.1: Model of the "ideal" video lecture usage process for students of a given course.

10.1.2 Event Data

The video lecture usage process is supported by several IT systems. The data related to student's personal information and their grades are registered in the university's information systems. The data related to video lecture views is supported the a third-party system named Mediasite. Such data has to be periodically extracted from both systems manually given technical and political difficulties (i.e., access permissions).

The data used in this case study relates to the academic years 2014-2015 and 2015-2016, and it contains **336.462** *video lecture views* and **159.134** *course grades* of **11.367** students, **8.893** video lectures and **1.750** courses.

The data is extracted from the different IT systems in the form of three data tables: *video lecture views*, *course grades* and *personal information* of students of the University. Each student and course has a unique identifier code (i.e., *student id, course code*). The data attributes available are described in Table 10.1.

Table 10.1: Event Data

(a) Student Data Attributes

Name	Description
Student Id	the anonymized unique ID of the student
Gender	the gender of the student
Nationality	the nationality of the student
Address	the postal code related to the student's registered address
Ed. Code	the code of the academic program in which the student is enrolled

(b) Videolecture View Data Attributes

Name	Description
Student Id	the anonymized unique ID of the student
Lecture Name	the name / number of the videolecture
Subject Code	the code of the course that provides the videolecture
Timestamp	the timestamp at the start of the view
Coverage	the time spent watching the videolecture (excluding pauses)
Duration	the time spent watching the videolecture (including pauses)
Quartile	the quartile in which the view happened
Academic Year	the academic year in which the view happened

(c) Course Grades Data Attributes

Name	Description
Student Id	the anonymized unique ID of the student
Subject Code	the code of the course
Date	the date of the exam
Quartile	the quartile in which the course was taken by the student
Academic Year	the academic year in which the course was taken by the student
Grade	the grade obtained by the student

These tables are merged into a single event data table using opportune joins between them. An anonymized fragment of the resulting data is presented in Table 10.2.

Table 10.2: A fragment of event data generated from the University's system: each row corresponds to an event.

Student Id	Nat.	Ed. Code	Course Code	Activity	Quartile	Acad. Year	Timestamp	Grade	...
1025	Dutch	BIS	2II05	Lecture 1	1	2014-2015	03/09/2012 12:05	6	...
1025	Dutch	BIS	2II05	Lecture 2	1	2014-2015	10/09/2012 23:15	6	...
1025	Dutch	BIS	1CV00	Lecture 10	3	2014-2015	02/03/2012 15:36	7	...
2220	Spanish	INF	1CV00	Lecture 1	3	2014-2015	20/03/2013 16:24	8	...
1025	Dutch	BIS	2II05	Exam	2	2014-2015	13/12/2012 12:00	6	...
2220	Spanish	INF	1CV00	Lecture 4	3	2014-2015	25/03/2013 11:12	8	...
2220	Spanish	INF	1CV00	Exam	3	2014-2015	04/04/2013 12:00	8	...
...	...	...	...	...	...	...	...	...	...

The data reveals enormous variability, e.g., thousands of students watch video lectures for thousands of courses and every course has a different set of video lectures, and they have different cultural and study backgrounds, which leads to different behavior. Therefore, we need to provide different reports and, within a report, we need to perform a comparative analysis of the students when varying the grade.

10.1.3 Analysis Purpose

The purpose of this case study is to show how raw data extracted from the University's IT systems can be transformed into reports that show insights about students' video lecture usage and its relation with course grades by using process mining, process cubes and analytic workflows.

The reports are composed of three sections: *course information*, *core statistics* and *advanced analytics*, as shown in Figure 10.2. An example report, where student information has been anonymized, can be downloaded from `https://www.dropbox.com/s/565zz94rdo6gg2r/report.zip?dl=0`. The analysis results refer to all students who registered for the course exam, independently whether or not they participated in it.

The course information section provides general information, such as the course name, the academic year, the number of students, etc. The core statistics section provides aggregate information about the students, such as their gender, nationality, enrolled bachelor or master program, along with course grades distribution and video lecture views. The advanced analytics section contains process-oriented diagnostics obtained through process mining techniques.

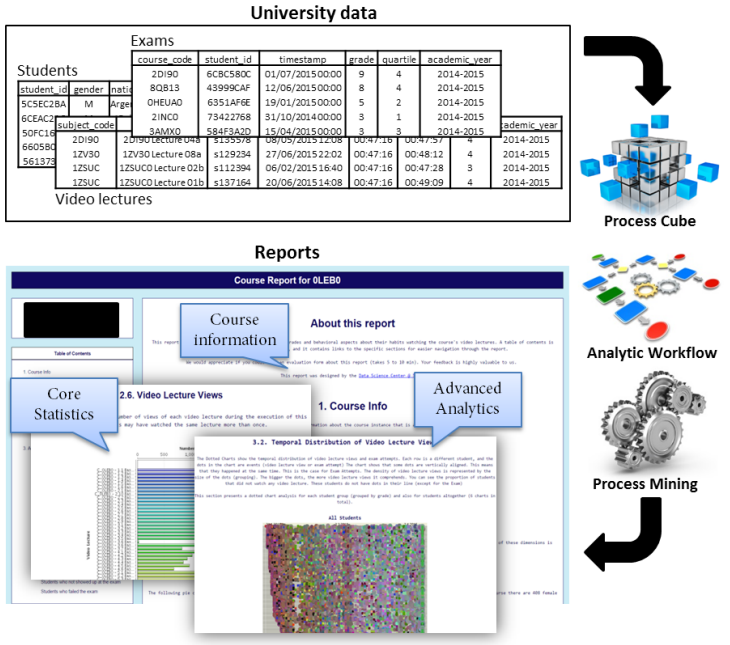


Figure 10.2: Overview of the case study: University data is transformed into reports by using process mining, process cubes and analytic workflows.

10.1.4 Related Work

This section discusses the related work done around business process reporting and educational data analysis. Related work about analytic workflows and process cubes is discussed in Chapters 4 and 3 respectively.

Business Process Reporting

Business Process Intelligence (BPI) is defined by [63] as the application of Business Intelligence (BI) techniques to business processes. However, behavioral properties of processes (e.g., control-flow) cannot be represented using traditional BI tools. Alternatively, Castellanos et al. [34] provides a broader definition: BPI exploits process information by providing the means for analyzing it to give companies a better understanding of how their business processes are ac-

tually executed. It incorporates not only metric-based process analysis, but also process discovery, monitoring and conformance checking techniques as possible ways to understand a process.

Business Process Reporting can be defined as the structured and periodical production of reports containing analysis of process data obtained through BPI techniques.

Business process management suites (e.g., SAP, Oracle) usually provide process reporting capabilities. Often, these process reporting capabilites are an adaptation of general-purpose reporting tools (e.g., Crystal Reports, Oracle Discoverer) to process data [63]. These general-purpose reporting tools are unable to analyze the data from a process perspective (e.g., discover a model).

Most process mining tools (e.g., ProM, Disco) are able to analyze from a process perspective, but they lack reporting capabilities. Others, such as Celonis, offer business process reporting capabilities. However, they are limited to only a few process-perspective analysis components, and each report instance has to be created manually. Furthermore, the event data used as input for the report can only be filtered from the original event data; the granularity level cannot be changed. Also, most of these tools do not allow the comparison of process variants (e.g., students with different grades).

Given the limitations described above, in this chapter we used a combination of process mining, analytic workflows and process cubes to provide fully automated process-oriented reports.

Educational Data Analysis

The analysis of educational data and the extraction of insights from it is related to two research communities: *Educational Data Mining* and *Learning Analytics*.

Educational data mining (EDM) is an emerging interdisciplinary research area that deals with the development of methods to explore data originating in an educational context. EDM uses computational approaches to analyze educational data in order to study educational questions [62, 129]. For example, knowledge discovered by EDM algorithms can be used not only to help teachers to manage their classes, understand their students' learning processes and reflect on their own teaching methods, but also to support a learner's reflections on the situation and provide feedback to learners. An extensive survey on the state of the art of EDM is presented in [129]

Learning analytics (LA) is defined by [137] as the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it

occurs. According to [54], the difference between EDM and LA is that they approach different challenges driving analytics research. EDM focuses on the technical challenge (e.g., How can we extract value from these big sets of learning-related data?), while LA focuses on the educational challenge (e.g., How can we optimise opportunities for online learning?). A discussion on the differences, similarities and collaboration opportunities between these two research communities is presented in [138].

Several process mining techniques (e.g., Fuzzy Miner [64]) have been applied successfully in the context of EDM [149] for analyzing study curriculums followed by students. Notably, the work introduced by [110] aims to obtaining better models (i.e., in terms of model *quality*) for higher education processes by performing data preprocessing and semantic log purging steps. However, most of these techniques are not suitable for analyzing video lecture usage by students, given the inherent lack of structure of such processes.

In this chapter, we will use existing and new process mining techniques to obtain insights of student behavior from an educational point of view.

10.2 Experiments

Before we generate a report for each course instance, we need to split the dataset first. For this purpose, we used the *Process Mining Cube* (PMC) tool introduced in Chapter 3. As mentioned before, the starting point is the event data obtained from multiple joins of the tables extracted from the University's systems (see Table 10.2 for an example fragment of the data)

Using the obtained event data, we created a process cube with the following dimensions: Student Id, Student Gender, Student Nationality, Student Education Code, Course Code, Activity, Grade, Timestamp, Quartile and Academic Year.

After *slicing* and *dicing* the cube, thousands of cells are produced: one for each combination of values of the "Course Code", "Quartile" and "Course Grade" dimensions. Each cell corresponds to an event log that can be analyzed using process mining techniques.

For each resulting cell of the cube, we created an automatically-generated report. Figure 10.3 represents the creation of the reports from event data as an abstract analysis scenario, based on the building blocks defined in Chapter 4.

For automatically generating the course reports we used RapidProM (introduced in Chapter 4), which extends the RapidMiner analytic workflow tool with process mining techniques. Figure 10.4a illustrates the analytic workflow that

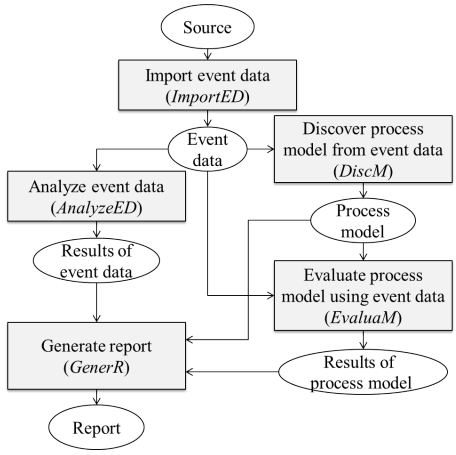


Figure 10.3: Abstract analysis scenario for generating reports from event data

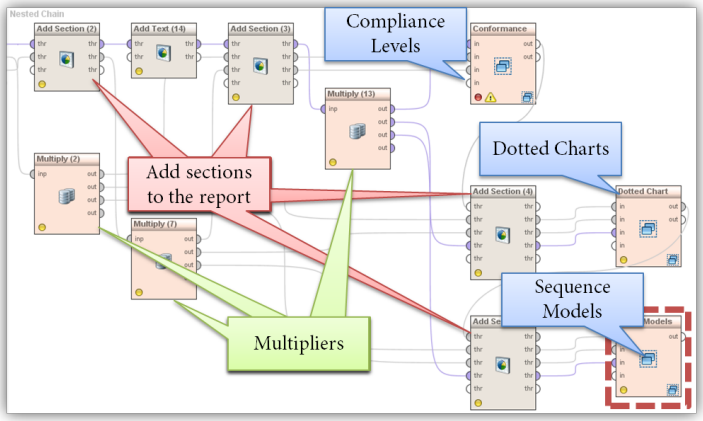
is used to generate each report. Figure 10.4b shows the explosion of the "Sequence Models" section of the analytic workflow.

The operators shown in Figure 10.4 are used for different purposes: *Multipliers* allow one to use the output of an operator as input for many operators. *Filter* operators select a subset of events based on defined criteria. *Process mining operators* are used to produce analysis results. For example, the operators highlighted in blue in Figure 10.4b produce a sequence model from each filtered event data.

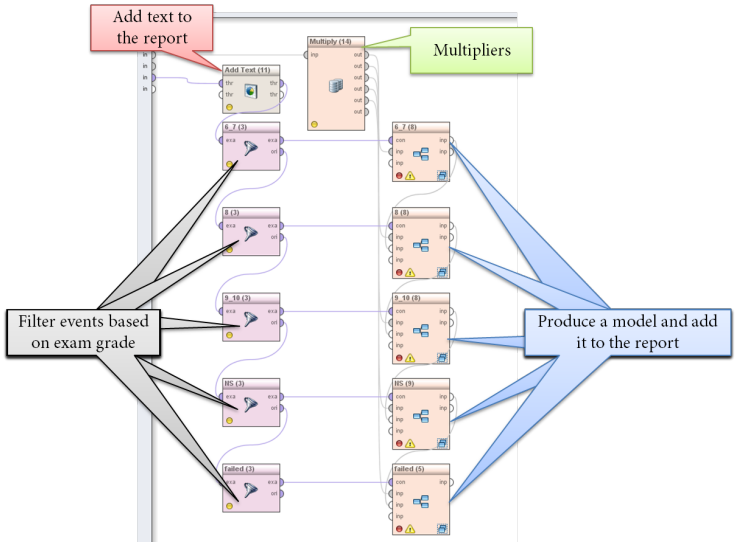
The complete RapidProM implementation of the analytic workflow used in this case study is available at `https://www.dropbox.com/s/g9spsziyv55vsro/single.zip?dl=0`. Readers can execute this workflow in RapidMiner to generate a report using the sample event log available at `https://www.dropbox.com/s/r3gczshxqxh6a6d/Sample.xes?dl=0`.[2]

We applied our approach that combines process mining, analytic workflows and process cubes to this case study in two evaluation rounds. The remainder of this section describe the work, reports, results and the feedback obtained on each round.

[2]When running the workflow, make sure that the *Read File* operator points to the sample event log and the "HTML output directory" parameter of the *Generate Report* operator points to the desired output folder.



(a) Advanced Analytics section sub-workflow



(b) Explosion of the “Sequence Models” sub-workflow

Figure 10.4: Implemented analytic workflow used to generate the reports. Each instance of a course can be automatically analyzed in this way resulting in the report described.

10.2.1 Initial Report

The first evaluation round was conducted in August 2015 and it used the event data corresponding to the academic year 2014-2015. The data used in this round contains **246.526** *video lecture views* and **110.056** *course grades* of **8.122** students, **8.437** video lectures and **1.750** courses. Concretely, we automatically generated a total of **8.750** course reports for 1750 courses given at the University in each of the 5 quartiles (i.e., 4 normal quartiles + *interim* quartile) of the academic year 2014-2015. For reliability of our analysis, we only selected the reports of courses where, on average, each student watched at least 3 video lectures. In total, 89 courses were selected and their reports were sent to the corresponding lecturers.

Section 10.2.1 shows the first report structure through an example of the reports sent to lecturers in this evaluation round. It also provides a detailed analysis of the findings that we could extract from the report for a particular course. Along with the report, we also sent an evaluation form to the lecturers. The purpose of the evaluation forms is to verify whether lecturers were able to correctly interpret the analysis contained in the report. The results obtained in the first evaluation round are discussed in Section 10.2.1.

Structure of the Report

To illustrate the structure, contents and value of the reports, we selected an example course: "Introduction to modeling - from problems to numbers and back" given in the third quartile of the academic year 2014-2015 by the Innovation Sciences department at the University. This course is compulsory for all first-year students from all programs at the University. In total, 1621 students attended this course in the period considered. This course is developed in a "flipped classroom" setting, where students watch online lectures containing the course topics and related contents, and in the classroom, they engage these topics in practical settings with the guidance of the instructor.

The video lectures provided for this course are mapped onto weeks (1 to 7). Within each week, video lectures are numbered to indicate the order in which students should watch them (i.e., 1.1 correspond to the first video lecture of the first week). As indicated by the course's lecturer, the first video lectures of each week contain the course topics for that week, and the last video lectures of each week contain complementary material (e.g., workshops, tutorials). The number of video lectures provided for each week depends on the week's topics and related activities, hence, it varies.

Students's behavior can be analyzed from many perspectives. As mentioned in Section 10.1.4, several process mining techniques have been applied in the context of educational data analysis [149].

Initially, we applied traditional process model discovery techniques (e.g., Fuzzy Miner [64], ILP Miner [170], Inductive Visual Miner [103]) to the educational data. However, given the unstructured nature of this data (i.e., students watching video lectures), the produced models were very complex (i.e., *spaghetti* or *flower* models) and did not provide clear insights. Therefore, we opted for other process mining techniques that could help us understand the behavior of students:

Figure 10.5.a shows for each video lecture the number of students that watched it. We can observe that the number of students that watch the video lectures decreases as the course develops: most students watched the video lectures corresponding to the first week (i.e., 1.X) but less than half of them watched the video lectures corresponding to the last week (i.e., 7.X). Note that within each week, students tend to watch the first video lectures (i.e., X.1, X.2) more than the last ones (i.e., X.5, X.6). This was discussed with the course's lecturer. It is explained by the fact that, as mentioned before, the first video lectures of each week contain the topics, and the last ones contain complementary material.

Figure 10.5.b shows for each student group (i.e., grouped by their grade) the level of conformance, averaged over all students in that group, of the real order in which students watch video lectures, compared with the "natural" or logical order, namely with watching them in sequence (i.e., from 1.1 to 7.4). The conformance level of each student is measured as the *replay fitness* of the data over a process model that contains only the "natural" sequential order. The replay fitness was calculated using conformance checking techniques [157]. We can observe that students with higher grades have higher levels of conformance than students with lower grades.

Figure 10.5.c shows the grade distribution for this course where each bar is composed by two parts corresponding to the number of students who watched at least one (red part) video lecture and the number of students who did not (blue part). We can observe that the best students (i.e., with a grade of 8 or above) use video lectures. On the other hand, we observe that watching video lectures does not guarantee that the student will pass the course, as shown in the columns of students that failed the course (i.e. grade ≤ 5).

Figure 10.6 shows dotted charts [140] highlighting the temporal distribution of video-lecture watching for two student groups: (a) students that failed the course with a grade of 5, and (b) students that passed the course with a grade of

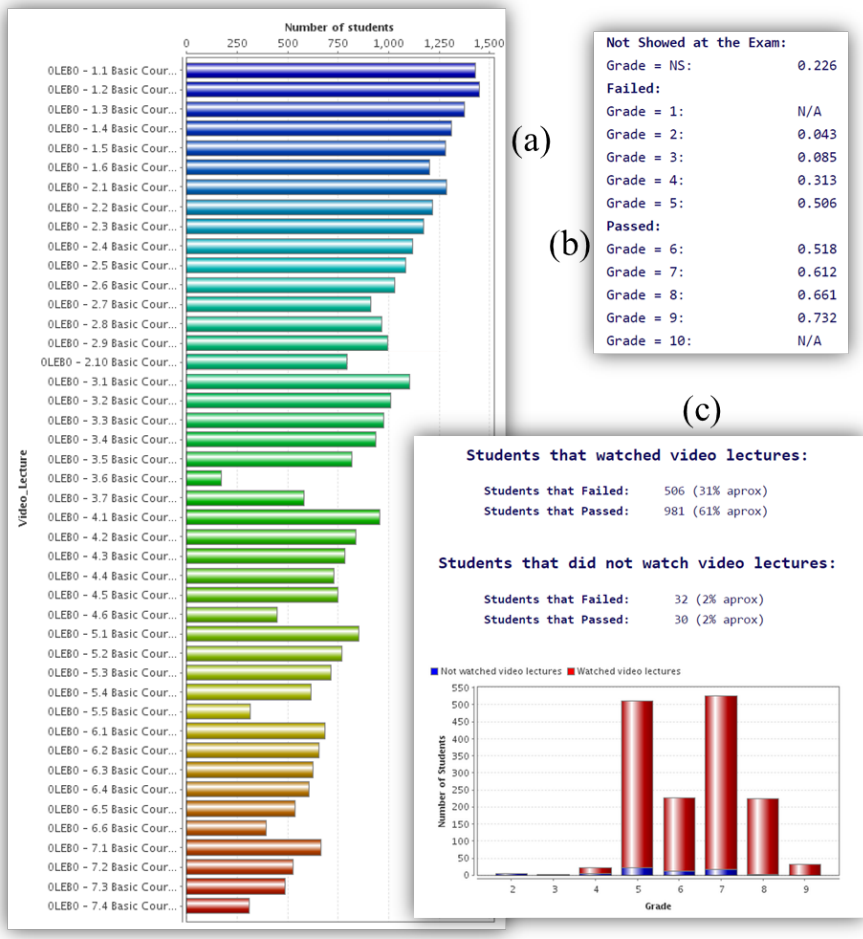
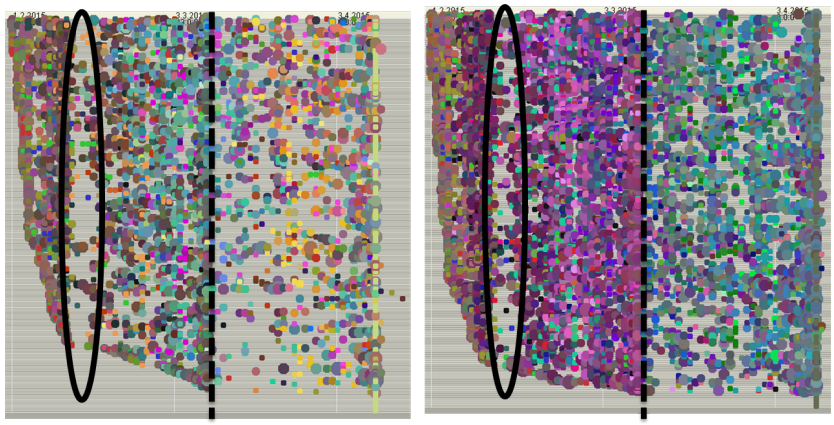


Figure 10.5: Analysis results contained in the report of the course 0LEB0:
(a) Number of students that watched each video lecture
(b) Conformance with the natural viewing order by course grade
(c) Grades distribution for students who watched video lectures (in red) or did not (in blue)

6 or 7. Each row corresponds to a student and each dot in a row represents that student watching a video lecture or taking the exam. Note that both charts show a gap where very few video lectures were watched, which is highlighted in the pictures through an oval. This gap coincides with the *Carnaval* holidays. We can observe that, in general, students that failed watched fewer video lectures. Also note that in Fig. 10.6.a the density of events heavily decreases after the mid-term exam (highlighted through a vertical dashed line). This could be explained by students being discouraged after a bad mid-term result. This phenomenon is also present in (b), but not equally evident. We can also observe that most students tend to constantly use video lectures. This is confirmed by the low number of students with only a few associated events.



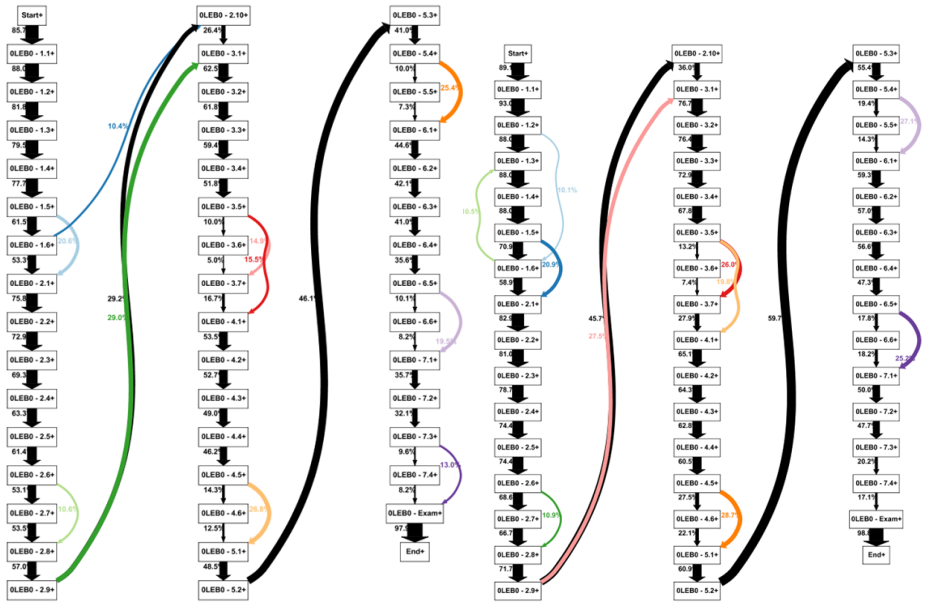
(a) Grade = 5 (failed) (b) Grade = 6 or 7 (passed)

Figure 10.6: Dotted charts for students grouped by their course grades

Figure 10.7 shows sequence analysis models that, given any ordered sequence of activities, reflects the frequency of directly-follows relations[3] as percentage annotations and as the thickness of edges. The highest deviations from the ordered sequence order are highlighted in colored edges (i.e., black edges correspond to the natural order). This technique was tailored for the generation of reports and it is implemented using a customized RapidProM extension. When comparing (a) students that passed the course with a grade of 6 or 7 with

[3]The frequency of directly-follows relations is defined for any pair of activities (A, B) as the ratio between the number of times that B is directly executed after A and the total number of times that A is executed.

(b) students that had a grade of 8 or 9, we can observe that both groups tend to make roughly the same deviations. Most of these deviations correspond to specific video lectures being skipped. These skipped video lectures correspond in most cases to complementary material. In general, one can observe that the thickness (i.e., frequency) of the arcs denoting the "natural" order (i.e., black arcs) is higher for (b), i.e., those with higher grades. Note that at the beginning of each week we can observe a *recovery* effect (i.e., the frequencies of the natural order tend to increase).



(a) Grade = 6 or 7 (b) Grade = 8 or 9

Figure 10.7: Sequence analysis for students grouped by their course grades.

Lecturers Evaluation

In addition to the qualitative analysis for some courses like such as the course analyzed in Section 10.2.1, we have also asked lecturers for feedback through

an evaluation form linked to each report.[4] The evaluation form provided 30 statements about the analysis contained in the reports (e.g., "Higher grades are associated with a higher proportion of students watching video lectures", "Video lecture views are evenly distributed throughout the course period"). Lecturers evaluated each statement on the basis of the conclusions that they could draw from the report. For each of the 30 statements, lecturers could decide if they agreed or disagreed with the statement, or, alternatively, indicate that they could not evaluate the statement (i.e., "I don't know").

In total, 24 of the 89 lecturers answered the evaluation form. Out of the 720 (24 x 30) possible statement evaluations, 437 statements were answered with "agree" or "disagree". The remaining cases in which the statement could not be evaluated can be explained by three possible causes: the statement is unclear, the analysis is not understandable, or the data shows no conclusive evidence.

In the case that a statement was evaluated with "agree" or "disagree", we compared the provided evaluation with our own interpretation of the same statement for that report and classified the response as *correct* or *incorrect*. In the case that a statement was not evaluated, the respose was classified as *unknown*.

Table 10.3 shows a summary of the response classification for each section of the report. In total, **89%** of the statement evaluations were classified as *correct*. This indicates that lecturers were capable to correctly interpret the analysis provided in the reports. Note that the *Conformance* section had the highest rate of *unknown* classifications (63.5%). This could be related to understandability issues of the analysis presented in that section.

Table 10.3: Summary of the classification of statement evaluations performed by lecturers

Statement Evaluation	Core Statistics Section	Advanced Analytics Section			**Sub Total**	**Total (%)**
		Conformance	Temp. Dist.	Seq. Analysis		
Correct	261	30	67	32	**390** (89%)	**61%**
Incorrect	28	5	8	6	**47** (11%)	
Unknown	95	61	69	58	**283**	**39%**

The evaluation form also contained a few general questions. One of such questions was: "Do you think this report satisfies its purpose, which is to provide insights about student behavior?", for which 7 lecturers answered "yes", 4 lecturers answered "no" and 13 lecturers answered "partially". All the lecturers

[4]The evaluation form is available at `https://www.dropbox.com/s/09y4ypklt70y6d9/form.zip?dl=0`

that responded "partially" provided written feedback indicating the improvements they would like to see in the report. Some of the related comments received were: "It would be very interesting to know if students: a) did NOT attend the lectures and did NOT watch the video lectures, b) did NOT attend the lectures, but DID watch the video lectures instead, c) did attend the lectures AND watch the video lectures too. This related to their grades", "The report itself gives too few insights/hides insights", "It is nice to see how many students use the video lectures. That information is fine for me and all I need to know", and "I would appreciate a written explanation together with your diagrams, next time". Another question in the evaluation form was: "Do you plan to introduce changes in the course's video lectures based on the insights provided by this report?", for which 4 lecturers answered "yes" and 20 answered "no". The results show that the analysis is generally perceived as useful, but that more actionable information is needed, such as face-to-face lecture attendance. However, this information is currently not being recorded by the TU/e. The feedback provided by lecturers was used to improve the report. These improvements are discussed in the next Section.

10.2.2 Final Report

We modified the reports based on the feedback obtained in the first evaluation round. The detail of the changes is presented in Section 10.2.2. To assess the quality of the improved report, we conducted a second evaluation round, which was conducted in March 2016 and it used the event data corresponding to the first two quartiles of the academic year 2015-2016. The data used in this round contains **89.936** *video lecture views* and **49.078** *course grades* of **10.152** students, **2.718** video lectures and **1.104** courses. Concretely, we automatically generated a total of **2.208** course reports for 1104 courses given at the University in each of the 2 first quartiles of the academic year 2015-2016. For reliability of our analysis, we only selected the reports of courses where, on average, each student watched at least 3 video lectures. In total, 56 courses were selected and their reports were sent to the corresponding lecturers.

Section 10.2.2 shows the changes introduced in the report based on the feedback obtained from lecturers in the first evaluation round. It also provides examples of the findings that several lecturers could extract from the report. Along with the report, we also sent an evaluation form to the lecturers. The purpose of the evaluation forms is to verify whether lecturers were able to correctly interpret the analysis contained in the improved report. Unfortunately, in this evaluation round no lecturer answered the evaluation form. Therefore, we

held face-to-face meetings with four lecturers, where the results included in the report were discussed. The insights obtained in these meetings are discussed in Section 10.2.2.

Changes in the Report

According to the feedback obtained from lecturers (reported in Table 10.3 in Section 10.2.1), the most problematic sections (i.e., highest rate of *unknown* classifications) were the Conformance (63.5% *unknown*) and Sequential Analysis (60.4% *unknown*) sections of the report (described in Section 10.2.1).

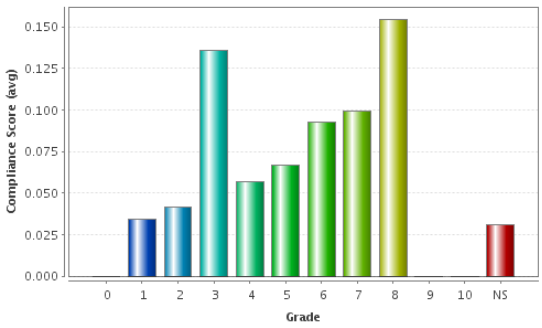
Given this feedback and the fact that the interpretation of a specific *replay fitness* value can be misleading for non-process-mining-experts, the conformance section was replaced for a section that describes the *compliance* of students with the "natural" order of watching video lectures based on simpler calculations, defined as follows.

Definition 10.1 (Compliance Score)**.** *For any given student, their compliance score (CS) w.r.t. the natural order is calculated as* $CS = \sum_{i=1}^{n-1} \frac{df(a_i, a_{i+1})}{count(a_i)}$*, where* $df(a_i, a_{i+1})$ *is the number of times that the student watched lecture* a_{i+1} *directly after* a_i*,* count(a_i) *is the number of times that the student watched the lecture* a_i *and* n *is the number of video lectures available for the course.*

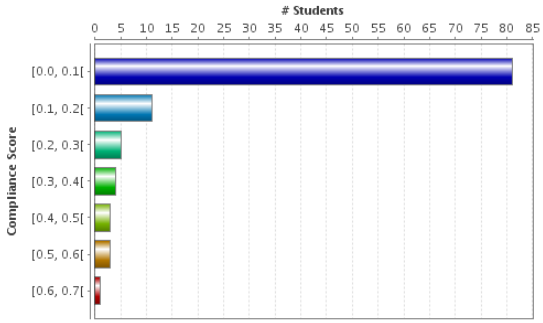
This new compliance score is easier to interpret: a value of X means that X percent of the video lectures watched by the student were watched in the natural order.

Figure 10.8 shows an example of the new compliance section of the report. It refers to the course "5ECC0 - Electronic circuits 2" (more details will be given later). Figure 10.8.a shows the average compliance scores according to the student's grades, while Figure 10.8.b shows the distribution of students according to their compliance scores.

Regarding the Sequence Analysis section, we simplified the explanatory text of this section in the report. However, this section presents inherent difficulties associated to the analysis of process models: most lecturers are not familiar with process models. Previously, sequence models only referred to frequency deviations. In this round, we decided to incorporate sequence models that show performance information. In these sequence models, an arc indicates the time between the start of the source activity (i.e., video lecture or exam) and the start of the target activity. From these models, one can observe if a given lecture is being fully watched, or if students are skipping most of it after watching



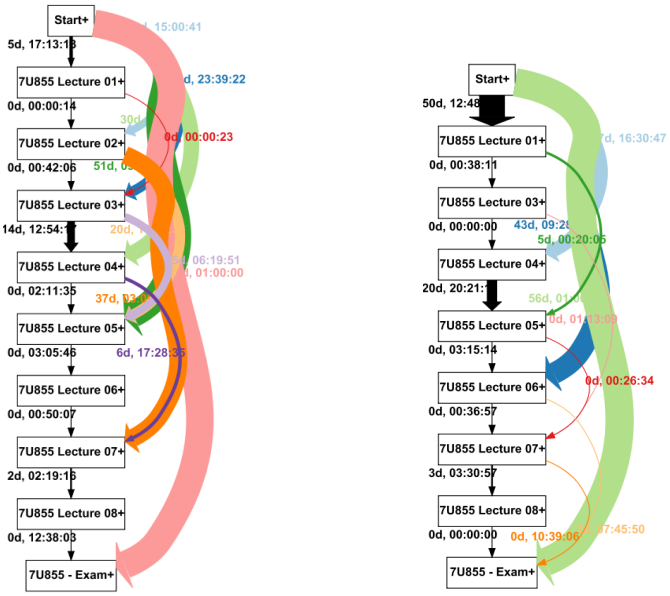
(a) Average student compliance with the "natural" order according to the student's grades



(b) Student distribution over compliance level by range.

Figure 10.8: New compliance section of the report for an example course (5ECC0 - Electronic circuits 2)

a few minutes. Figure 10.9 shows an example of a sequence model annotated with performance information. This model was obtained from one of the course reports (7U855 - Research methods for the built environment) sent in this evaluation round. From these models we can get interesting insights about the students' behavior on this course. For example, in Figure 10.9.a (i.e., students that obtained a 6 or a 7 in the exam) the arrow between *Lecture 01* and *Lecture 02* states that students that watched *Lecture 02* directly after *Lecture 01*, started watching *Lecture 02* 14 seconds (in average) after started watching *Lecture 01*. However, in Figure 10.9.b (i.e., students that obtained a 8, 9 or 10 in the exam) this specific behavior is not observed.



(a) Grade = 6 or 7. (b) Grade > 7.

Figure 10.9: Sequence models annotated with performance information for students grouped by their grade. The models were obtained from the report of course 7U855 - Research methods for the built environment.

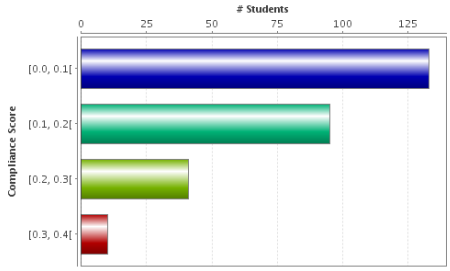
Lecturers Evaluation

As mentioned before, from the 56 reports sent to lecturers in this evaluation round, we obtained no responses to the corresponding evaluation forms. Therefore, we held face-to-face meetings with four lecturers from different departments of the University to discuss the report in general, and to evaluate if the changes introduced in this evaluation round did actually improve the understandability of the report.

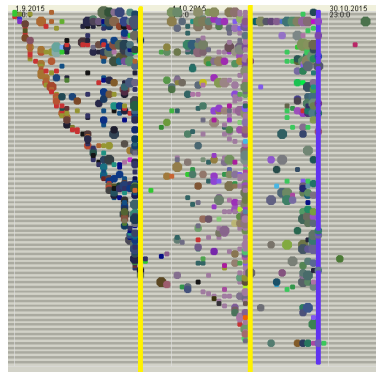
In the remainder of this section, we summarize the insights obtained by lecturers when discussing the reports in the face-to-face meetings.

The first lecturer we met was responsible for the course 1CV00 - Deterministic Operations Management, provided by the Industrial Engineering department. In this course, lectures are grouped by topic (i.e., 2 lectures per topic) and topics are independent from each other. Figure 10.10.a shows the distri-

bution of students according to their compliance scores for this course. In this chart, we can observe that students have a very low compliance score in general, and it had no correlation with grades. The lecturer defined this behavior as "expected" since the course topics are independent. Figure 10.10.b shows the dotted chart containing all the students of the course. Here we can observe two peaks of video lecture usage in weeks 4 and 7 (highlighted with vertical yellow lines), but without context information, we cannot explain why they happened. The lecturer immediately identified these two peaks as the two *mid-term* exams that are part of the course. The interpretation given by the lecturer was that students were using the video lectures to study for these exams. This behavior was expected by the lecturer, but in the past he did not have the information to either confirm or deny it.



(a) Student distribution over compliance level by range. Compliance scores are relatively low.

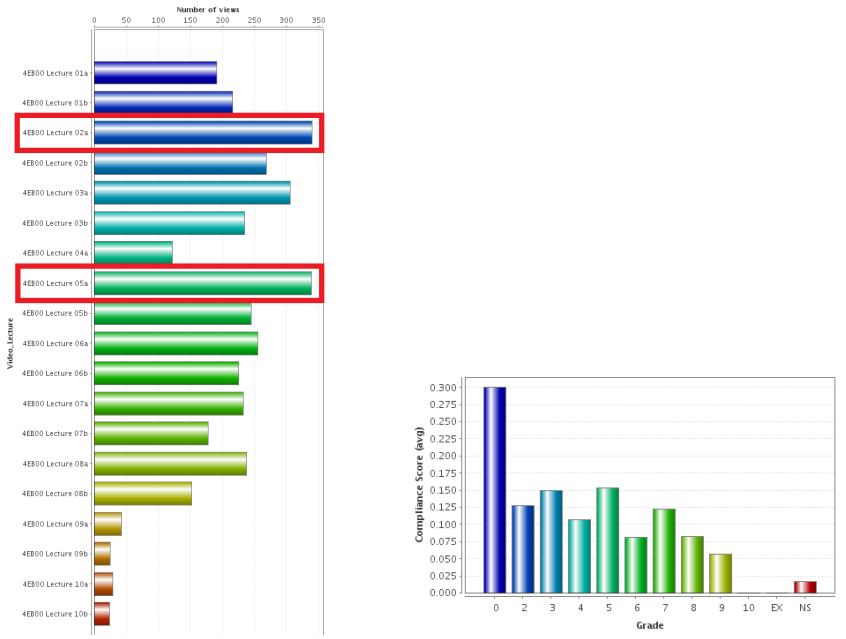


(b) Dotted chart for all the students enrolled in the course. Video lecture usage peaks are highlighted with yellow vertical lines.

Figure 10.10: Analysis results included in the report of the course 1CV00.

The second lecturer was responsible for the course 4EB00 - Thermodynamics, provided by the Mechanical Engineering department. In this course, some topics build on top of knowledge acquired in previous topics, but others are independent. Figure 10.11.a shows, for each lecture, the total number of views. We can observe that *Lecture 02a* and *Lecture 05a* had the highest number of views. The lecturer determined that this behavior was expected, since *Lecture 02a* contained most of the definitions and knowledge that students needed to "remember" from previous courses. On the other hand, *Lecture 05a* was related to *Entropy*, which was the most difficult topic of the course for students.

Figure 10.11.b shows the average student compliance with the "natural" order according to the student's grades. We can observe that there is a negative correlation between the compliance scores and the grades. According to the lecturer: "A possible explanation of this could be that students with bad grades could have skipped face-to-face lectures and then needed to watch all the video lectures, while good students attended face-to-face lectures and only watched some video lectures if they needed to clarify something".



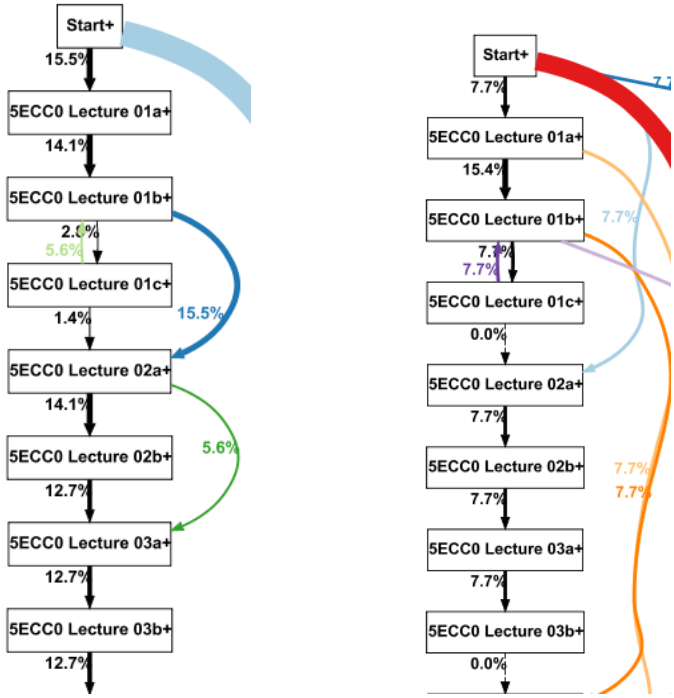
(a) Number of views of each video lecture of the course. Lectures 02a and 05a were the most watched by students (highlighted in red).

(b) Average student compliance with the "natural" order according to the student's grades. There seems to be a negative correlation between compliance scores and grades.

Figure 10.11: Analysis results included in the report of the course 4EB00.

The third lecturer was responsible for the course 5ECC0 - Electronic Circuits 2, provided by the Electrical Engineering department. In this course, all the topics were related, every topic built-up on the previous one. Figure 10.8.a showed the average student compliance with the "natural" order according to the student's grades. We can observe a positive correlation between compliance

scores and grades. The lecturer was positively surprised by this finding, but he considered that the correlation was not strong. Figure 10.12 shows a fragment of the sequence model with frequency deviations for two different groups of students of the course (i.e., those with a grade lower than 5, and those with a grade equal to 6 or 7). We can observe in Figure 10.12.a that *Lecture 01c* is being skipped by 13% of the students that watched the *Lecture 01b*. This behavior does not occur for students with higher grades (shown in Figure 10.12.b). The lecturer then considered this finding as "unexpected, but positive", since *Lecture 01c* consists of the basic topics from the previous course (i.e., Electronic Circuits 1) and it was meant to refresh student's knowledge. According to the lecturer, the fact that students did not need to watch it is positive.



(a) Grade < 5. (b) Grade = 6 or 7.

Figure 10.12: Fragment of the sequence model with frequency deviations for all students. In (a), Lecture 1c is being skipped. These charts were included in the report of the course 5ECC0 - Electronic Circuits 2.

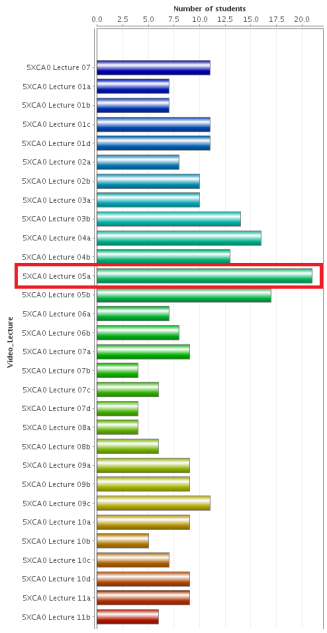
The fourth lecturer was responsible for the course 5XCA0 - Fundamentals of Electronics, provided by the Electrical Engineering department. This course considers topics are relatively independent from each other. Figure 10.13.a shows, for each lecture, the total number of views. It is interesting to notice that the *Lecture 05a*, the video lecture most watched by students (highlighted in red) is an *instruction* lecture (i.e., a lecture that consists of exercises instead of topics). Given this finding, the lecturer has expressed the intention of splitting that video lecture, for the next executions of the course, into a series of 10-minute web lectures comprehending all the different types of exercises covered in the video lecture. Figure 10.13.b shows the student distribution over ranges of compliance score. It is clear from the chart that most students have a very low compliance score w.r.t. the "natural" viewing order. This was justified by the lecturer through the following statement: "The topics are relatively disconnected, and it seems that most students would watch only specific lectures".

The general comments that we received from lecturers are summarized as follows. "It would be interesting to see the correlation with face-to-face lectures to see if students use video lectures as a replacement or as a complement for them". "Video lectures are very good for the *middle* students. good students do not seem to need them as much". "I should split the most visited video lectures into a series of web lectures (i.e., 10 minute recordings of specific topics) so I could really know which topics are the most difficult for the students". "Students tend to use exercise lectures much more intensively than the actual theory. They seem to be exam-oriented, as they prepare mostly watching exercises".

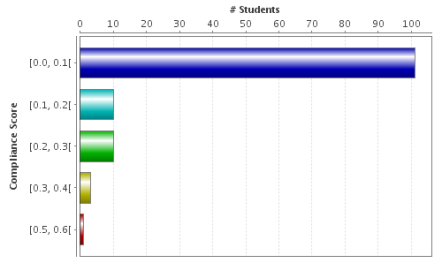
Regarding the report itself, again we received suggestions to incorporate face-to-face lecture attendance. As mentioned in Section 10.2.1, it is very difficult to record face-to-face attendance of students for technical reasons. Other lecturers suggested that we incorporate student feedback into the report. We certainly recognize the potential that incorporating the students' feedback on the report could have in the insights that the lecturer can obtain from it. We plan to incorporate this feedback and its potentially positive effects in the reports for the next quartile.

10.3 Conclusion

This chapter has illustrated the benefits of combining the complementary approaches of process cubes and analytic workflows in the field of process mining. In particular, the combination is beneficial when process mining techniques need to be applied on large, heterogenous event data of multidimensional na-



(a) Number of views of each video lecture of the course. Lecture 05a was the most watched by students (highlighted in red).



(b) Student distribution over compliance level by range. Most students have a very low compliance score (i.e., between 0% and 10%).

Figure 10.13: Analysis results included in the report of the course 5XCA0.

ture.

To demonstrate such benefits, we applied the combined approach in a large scale case study where we provide reports for lecturers. These reports correlate the grades of students with their behavior while watching the available video lectures. We evaluated the usefulness of the reports in two evaluation rounds. The second evaluation round presented an improved report, which was modified based on the feedback obtained in the first evaluation round. Unlike existing *Learning Analytics* approaches, we focus on dynamic student behavior. Also, descriptive analytics would not achieve similar analysis results because they do not consider the process perspective, such as the ordering of watching video lectures.

Educational data has been analyzed by some disciplines in order to under-

stand and improve the learning processes [54, 62, 129, 137, 138], even employing process cubes [161]. However, these analyses were mostly focused on individual courses. No research work has previously been conducted to allow large-scale process mining analysis where reports are automatically generated for any number of courses. Our approach has made it possible by integrating process mining with analytic workflows, which have been devised for large-scale analysis, and process cubes, which provide the capabilities needed to perform comparative analyses.

As future work, the report generation will be extended to *Massive Open Online Courses* (MOOCs) given by Eindhoven University of Technology. This type of courses are particularly interesting due to the fact that face-to-face lectures are not used: video lectures are the main channel used by students for accessing the course topics. For example, over **100.000** people from all over the world registered for the first two executions of the MOOC *Process Mining: Data science in Action*.[5] We also plan to apply this analysis to the courses provided by the *European Data Science Academy* (EDSA).[6]

[5] `http://www.coursera.org/course/procmin`
[6] `http://edsa-project.eu`

Chapter 11

Comparative Analysis of Business Process Outsourcing Services

*Disclaimer: The data used in this chapter is subject to a non-disclosure agreement signed by the author and by **Xerox Services**. Therefore, the data is not publicly available.*

Business process outsourcing (BPO) is the contracting of business activities and functions (usually non-primary) to a third-party provider. BPO services usually include payroll, human resources (HR), accounting and customer/call center relations.

BPO organizations tend to specialize in providing services in a large-scale e.g., to many different companies. In such a way, they can benefit from larger-scale operations by reducing their marginal costs. However, given the different nature of their clients and their processes, usually BPO organizations have to adapt their services to fit different requirements. This leads to a natural variability in the way that such services (i.e., business processes) are executed. Process variants may manifest due to differences in the nature of clients, local regulations, type and urgency of cases, SLAs, etc. It is crucial for BPO organizations to gain insights on such process variants in order to learn from best-practices, find root causes for inefficiencies, and improve their competitiveness in the global

market.

In this chapter, we use the tools and techniques proposed in Part II to perform a comparative analysis of a digitalization service process for several types of documents in a BPO organization, considering multiple perspectives such as control-flow and performance.

The remainder or this chapter is structured as follows. Section 11.1 introduces the context of this case study. Section 11.2 describes the experiments performed, and the obtained results. Section 11.3 discusses the results and their impact on the company. Finally, section 11.4 concludes the chapter.

11.1 Context

Xerox Corporation is a Fortune-500 USA-based global corporation that sells print and digital document products and services in more than 160 countries. In 2016, Xerox Corporation separated its *Xerox Services* division that handled all business process service operations into a new company named *Conduent*, which has now over 85,000 employees in more than 40 countries. One of the business process service operations provided by Xerox Services is *business process outsourcing*.

One of the many BPO services offered by Xerox Services (currently by Conduent) is the digitalization of paper-printed health insurance forms, which usually include both handwriting and printed information.

The remainder of this section describes the document digitalization process, the dataset used, and the purpose of the analysis in the form of process questions.

11.1.1 Process Description

This case study relates to the transaction processing business unit within Xerox Services. More specifically, we analyzed the process pertaining to the data entry back-office operations of insurance claim forms.

Figure 11.1 shows two examples of such forms, submitted in paper by a US healthcare-related organization, that are digitalized by Xerox Services.

Forms submitted to and by the insurance providers need to be digitized before the claims can be processed. *Business Process Outsourcing* (BPO) organizations such as Xerox Services assist the insurance providers in this process. In the digitalization process, each *case* refers to the data entry operations of one

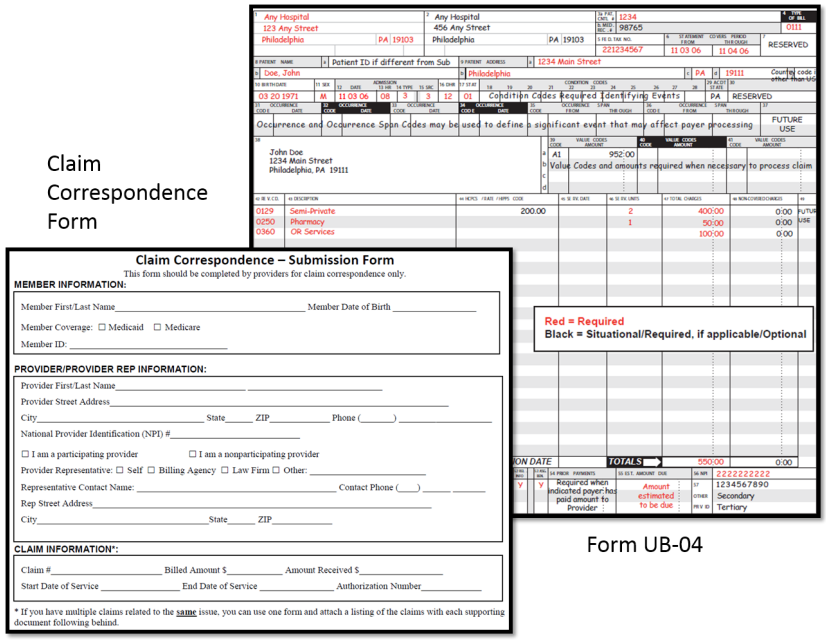


Figure 11.1: Example of two paper-printed forms digitalized by Xerox Services. The UB-04 (on the right) form is a claim form used by hospitals, nursing facilities, in-patient, and other facility providers. The correspondence claim form (on the left) defines a request for additional information in order for a claim to be considered clean, to be processed correctly or for a payment determination to be made.

instance of an insurance claim form. Xerox Services handles the data entry operations of millions of insurance claim forms per month.

Forms received by Xerox Services are classified and sorted depending on the type of form e.g., HCFA, UB04, Dental, correspondence claim, etc. More fine-grained classifications further refining each type are possible (e.g., HCFA standard, HCFA careplus, etc), thereby defining a taxonomy. Different classes in this taxonomy are divided into so-called *batches*. Each batch relates to a set of forms of one specific type (e.g., HCFA standard) to be processed. Each type of form is handled differently: many steps involved in the processing of a form may not be performed in the handling of other forms.

Figure 11.2 shows a process model generated with the Disco tool that represents the control-flow of the process based on its observed executions including different batches related to different forms. The unreadability and complexity

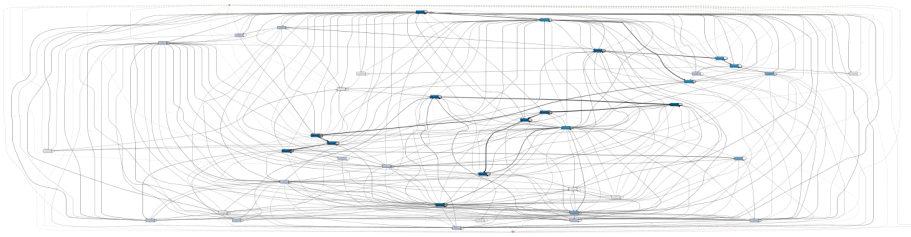


Figure 11.2: Process Model that represents all the behavior included in the event data related to different batches.

of this model further confirms the need to analyze the variability within the process.

11.1.2 Event Data

Xerox handles millions of transactions every day. In this chapter, we only consider the transactions of one month (i.e., November 2015) related to one US-based client. This raw data obtained from Xerox contains information on 94 different batches (i.e., form types) and contains more than 40 million rows of data that can be related to events in the process. Table 11.1 shows an anonymized extract of the data obtained from Xerox.

The *DCN* attribute is related to a particular form being processed. Hence, it indicates the case ID of the process. The *BATCHNAME* attribute indicates the batch (i.e., type of insurance claim form) of the form being processed. The *TASK* attribute refers to the activity being executed. The *START* and *STOP* attributes indicate the moment when each avtivity was started and completed. Finally, the *USER* and *LOCATION* attributes indicate the resource that is executing the activity and its location.

It is important to note that the process is heavily automatized, and humans are seldom involved in its execution. For example, the resource “MrAuto” (the most frequent one) indicates that the activity was performed automatically by a system. Other resources with an id number (e.g., 30192788) correspond to humans, which are usually related to very specialized and infrequent tasks such as visual confirmations and checks.

Table 11.1: A fragment of raw data generated by Xerox's systems

BATCHNAME	DCN	TASK	START	STOP	USER	LOCATION
BATCH_1	151102000076	OCRFlowValve	02/11/2015 11:27:46	02/11/2015 11:27:47	MrAuto	Kochi
BATCH_1	151102000076	Images2Humana	02/11/2015 11:44:29	02/11/2015 11:44:32	MrMQClaims	Kochi
BATCH_1	151102602347	OCRFlowValve	02/11/2015 13:01:04	02/11/2015 13:01:04	MrAuto	Domestic
BATCH_1	151102602348	OCRFlowValve	02/11/2015 13:01:04	02/11/2015 13:01:04	MrAuto	Domestic
BATCH_1	151102602347	Images2Humana	02/11/2015 13:14:43	02/11/2015 13:14:46	MrMQClaims	Domestic
BATCH_1	151102602348	Images2Humana	02/11/2015 13:14:43	02/11/2015 13:14:46	MrMQClaims	Domestic
BATCH_1	151102000076	KeyFlowValve	03/11/2015 01:15:30	03/11/2015 01:15:30	MrAuto	Kochi
...	...	...	...	...	...	...
BATCH_6	151106807336	DomesticRouter	07/11/2015 00:56:10	07/11/2015 00:56:11	MrAuto	Cebu
BATCH_6	151106807336	ToCebu	07/11/2015 00:57:10	07/11/2015 00:57:16	MrAuto	Cebu
BATCH_6	151106807336	EnrollmentTracker	07/11/2015 06:58:57	07/11/2015 06:58:58	MrAuto	Cebu
BATCH_6	151106807336	OCRReview	07/11/2015 07:11:58	07/11/2015 07:11:58	MrAuto	Cebu
...	...	...	...	...	...	...
BATCH_11	151106010641	QIMiner	10/11/2015 21:33:59	10/11/2015 21:34:06	MrAuto	India
BATCH_11	151106010641	XMLToX12	10/11/2015 21:36:26	10/11/2015 21:36:28	ClaimsX12	India
BATCH_11	151106010641	Transmit	10/11/2015 21:37:14	10/11/2015 21:37:19	MrDentalBind	India
BATCH_11	151106010641	TransmitACK	10/11/2015 21:37:22	10/11/2015 21:37:22	MrX12Ack	India
BATCH_11	151106010641	Verify	11/11/2015 07:29:00	11/11/2015 07:35:00	30212621	India
BATCH_11	151106010641	Corrects	11/11/2015 07:57:00	11/11/2015 07:59:00	30192788	India
...	...	...	...	...	...	...

Also note that a row of this raw data does not necessarily correspond to a single event, but can be related to multiple events e.g., a single row can be related to two events (i.e., a "start" and an "end" event) using the its *START* and *END* attributes).

11.1.3 Analysis Purpose

Xerox is interested in analyzing the processes followed across different batches and wants to obtain usable insights on their executions. To obtains such insights, we analyzed the data presented above in order to answer the following research questions:

PQ1: What are the workflows followed by the different batches, what do they have in common, and where do they differ?

PQ2: Can we come up with a standardized process model that covers all batches?

PQ3: Are there particular paths that influence process performance, and are these paths common across the batches?

In the next section we will address how to answer each one of these questions in detail.

11.2 Experiments

This section reports on the experiments performed in order to answer each of the four process questions asked by Xerox.

The first thing to notice about this dataset is the large amount of different batches (i.e., 94 different batches). Performing a pair-wise comparison of these 94 batches leads to 4.371 different combinations. Many of these batches are very different from each other, as they correspond to completely different forms. Comparing very different batches will result in obvious differences, and will not provide usable insights. Not all pair-wise comparisons are relevant for the analysis purpose presented above. Hence, we will focus on detecting interesting comparisons (i.e., batches that are similar to each other) in order to reduce the number of combinations to analyze.

Several pre-processing steps are performed in order to focus on some of the batches. Then, such batches are compared to each other so that we can obtain actionable and useful insights in order to answer the research questions described in Section 11.1.3.

Figure 11.3 illustrates the experimental design including all the steps performed in this experiment:

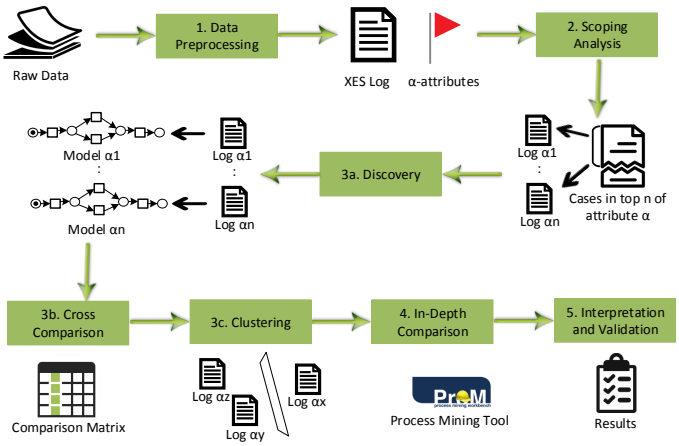


Figure 11.3: Experimental design: steps included in the experiments over Xerox data

These steps can be grouped into three phases:

1. **Data preparation and scoping:** This phase corresponds to steps 1 and 2 in Figure 11.3.

   Input: the raw data as shown in Section 11.1.2.

   Output: a set of all the event logs corresponding to the most frequent batches.

2. **Identification of interesting batch comparisons:** This phase corresponds to steps 3a, 3b and 3c in Figure 11.3.

   Input: a set of event logs corresponding to the most frequent batches.

   Output: a clustering over the set of event logs corresponding to the most frequent batches. Each cluster contains batches that are similar to each other.

3. **In-depth batch comparison:** This phase corresponds to steps 4 and 5 in Figure 11.3.

   Input: a set of clusters containing batches that are similar to each other.

   Output: answers to the research questions described in Section 11.1.3

The execution of the first two phases is supported by the use of a *process mining workflow* (introduced in Chapter 4) that automatically performs tasks such as data processing, analysis scoping and batch clustering, as shown in Figure 11.4.

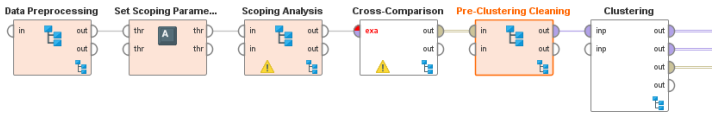


Figure 11.4: The RapidProM workflow used for the first two phases of the experiment

The execution of the third phase is supported by the use of the *process comparator tool* (introduced in Chapter 5)

The concrete execution of each phase and its steps are presented as follows.

11.2.1 Data Preparation and Scoping

As mentioned earlier, we can divide this phase into two steps: data preparation and scoping analysis (i.e., steps 1 and 2 in Figure 11.3).

In the *data preparation* step, we performed several data transformations on the raw input data (described in Section 11.1.2) These concrete transformations were implemented as the process mining (sub) workflow shown in Figure 11.5, and are described as follows:

1. We enriched the set of attributes based on the research questions. Since we are interested in analyzing different batches, we set the attribute `BATCHNAME` as the α attribute to be used in the comparison process.

2. We refined the raw input data into event level data. Each row in the raw input data includes two timestamps (i.e., start and end), therefore we divided each row into two events based on that (i.e., one start event and one end event).

3. We removed incomplete cases from the data. Based on statistics on the start and end activities for all cases, we removed the ones that have start or end activities that are not frequently observed. In this experiment, we removed 318,002 cases. The resulting data contains a total of 936,720 cases and 31,660,750 events.

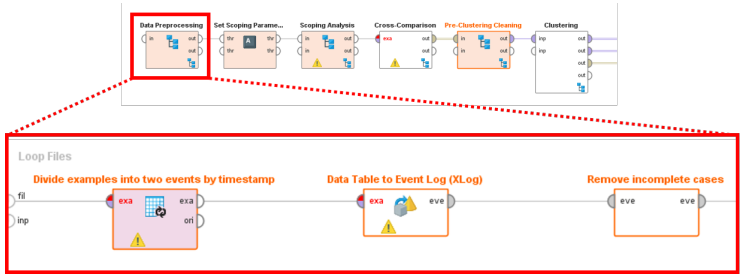


Figure 11.5: The RapidProM (sub) workflow used for the data preparation step.

As a result of this step, the raw input data was transformed into usable event data.

In the *scoping analysis* step, we took the usable event data obtained from the previous step, and we filtered out the infrequent batches based on their

frequency of occurrence (i.e., number of traces). The scoping analysis step was implemented as the process mining (sub) workflow shown in Figure 11.6.

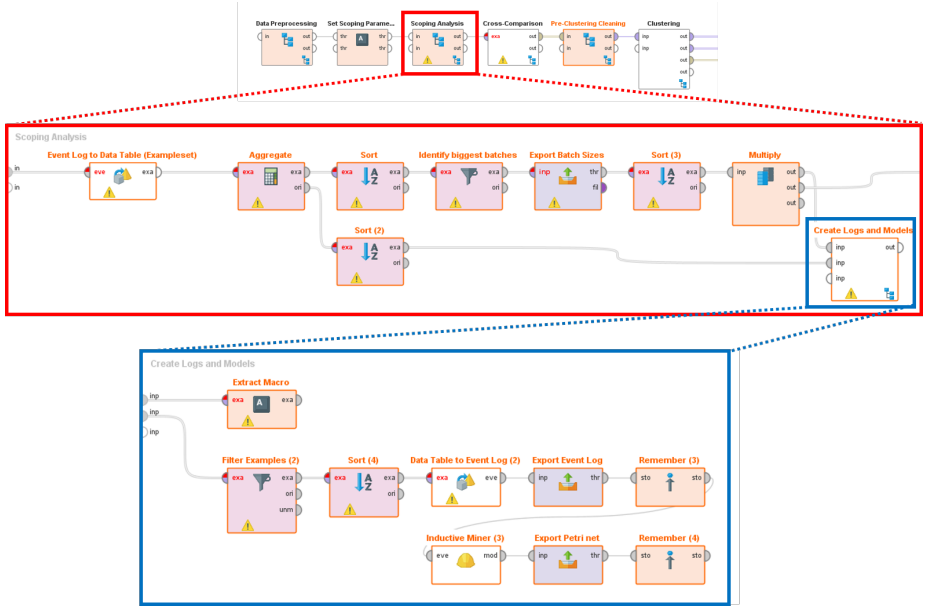


Figure 11.6: The RapidProM (sub) workflow used for the scoping analysis step.

We first aggregated all the events based on their `BATCHNAME` values. Then, we filtered the popular batches based on their occurrence frequency. Note that there are 94 different batches in our log. However, in order to scope the analysis into a feasible experiment, we selected the 10 most frequent ones. Their corresponding batch identifiers are 1, 4, 2, 11, 7, 18, 3, 58, 23, and 30 respectively, each having between 424,560 and 8,684,476 cases, as shown in Table 11.2.

As a result of this step, we selected the event data corresponding to each of the ten most frequent batches batch and transformed it into ten event logs (i.e., one per batch).

11.2.2 Identification of Interesting Batch Comparisons

Given the set of event logs related to the most frequent batches obtained from previous phase, the next phase is to identify interesting batch comparisons. To

Table 11.2: The 10 most frequent batches in the data

Batch #	number of cases
BATCH_1	8.684.476
BATCH_4	6.886.488
BATCH_2	4.184.606
BATCH_11	2.339.760
BATCH_7	2.054.370
BATCH_18	1.151.046
BATCH_3	1.002.792
BATCH_58	549.328
BATCH_23	476.996
BATCH_30	424.560

do this, we need to cluster batches based on their similarity. In this way we can avoid comparing notoriously-different batches and focus only on those that are similar to each other in order to analyze them and learn from their differences. As mentioned earlier, this phase corresponds to steps 3a (i.e., discovery), 3b (i.e., cross-comparison) and 3c (i.e., clustering) as shown in Figure 11.3.

Figure 11.7 shows the process mining (sub) workflow that implements the discovery and cross-comparison steps of this phase (i.e., steps 3a and 3b).

The purpose of these two steps is to compare the selected batches based on the analysis of their high-level process models. These models can be retrieved by process discovery techniques. There are several well-known discovery algorithms in literature, such as the Alpha miner [167], ILP miner [170] and Inductive Miner [102]. Considering the amount of events in our logs as well as the quality of the discovered processes (e.g., soundness and fitness), we have chosen the Inductive miner in the implementation of the discovery step of this phase (i.e., step 3a). Besides the fact that the Inductive Miner is the state-of-the-art process discovery, other techniques incline to produce models that are unable to replay the log well, create erroneous models, or have excessive run times. The output of step 3a is a collection of process models: one for each selected batch.

In step 3b, we used the *cross-conformance* technique (i.e., conformance checking of a model with respect to a different event log) in order to compare the batches. This concept is introduced in [28], where the so-called *comparison table* is presented. A comparison table consists of rows (i.e., event logs), columns

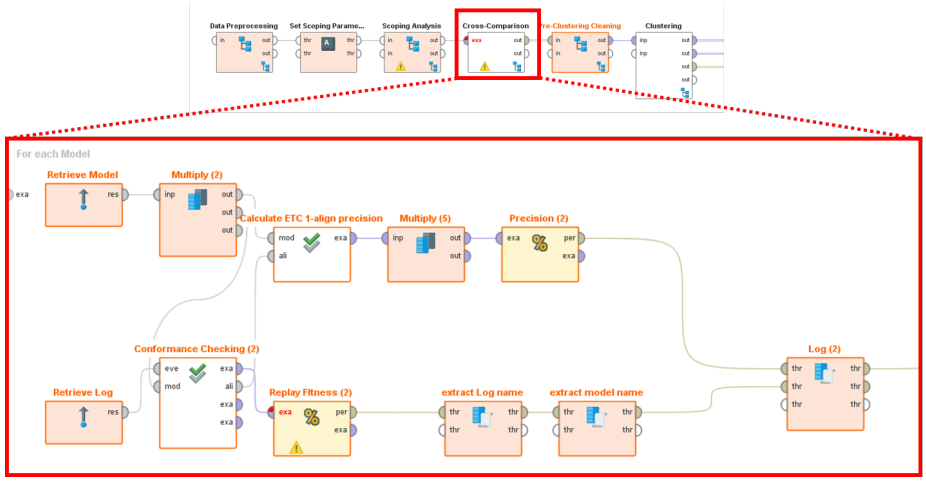


Figure 11.7: The RapidProM (sub) workflow used in this phase for steps 3a (i.e., discovery) and 3b (i.e., cross-comparison) of the experimental design.

(i.e., process models) and cells contain metrics. In this table, there are three types of metrics, namely *process model metrics, event log metrics,* and *comparison metrics*. Process model metrics are metrics calculated using only the process model, such as total number of nodes in the process model, cyclicity, or concurrency in the process model. Event log metrics are metrics calculated based on the event logs only, such as the total number of traces and events, average trace duration, etc. Finally, comparison metrics are calculated using both event logs and process models. They are used to compare modeled and observed behavior and they include metrics such as fitness, precision, generalization, and simplicity.

In this experiment, we used the *fitness* comparison metric to measure the conformance and cross-conformance of the sub-logs (i.e., batches) and the discovered models obtained from the previous step. We choose fitness rather than the other metrics due to the need of Xerox Services to have process models which allow for most of the observed behavior.

Table 11.3 shows the results of the cross comparison step (using the fitness metric) between logs and models related to the selected batches. Each row represents a log of a particular batch n (Log_n), and each column represents a discovered model from a particular batch m ($Model_m$). Each cell contains the

fitness value after replaying a log into a process model.

Table 11.3: Comparison table showing the comparison metric (i.e., fitness) between logs and models of the selected batches. Cell **(x,y)** indicates the replay fitness of the event log related to batch **x** with respect to the process model related to batch **y**.

	Model_1	Model_2	Model_3	Model_4	Model_7	Model_{11}	Model_{18}	Model_{23}	Model_{30}	Model_{58}
Log_1	0.71	0.71	0.41	0.35	0.56	0.56	0.42	0.36	0.38	0.61
Log_2	0.63	0.62	0.40	0.22	0.46	0.45	0.41	0.23	0.16	0.48
Log_3	0.39	0.39	0.79	0.56	0.40	0.40	0.79	0.59	0.41	0.30
Log_4	0.26	0.26	0.42	0.65	0.26	0.26	0.42	0.66	0.33	0.22
Log_7	0.58	0.59	0.46	0.36	0.69	0.69	0.46	0.39	0.44	0.51
Log_{11}	0.48	0.48	0.18	0.25	0.60	0.61	0.18	0.26	0.47	0.38
Log_{18}	0.37	0.37	0.71	0.48	0.40	0.40	0.71	0.51	0.47	0.30
Log_{23}	0.26	0.26	0.42	0.63	0.26	0.26	0.42	0.66	0.27	0.22
Log_{30}	0.24	0.24	0.29	0.26	0.24	0.23	0.29	0.27	0.66	0.31
Log_{58}	0.55	0.55	0.26	0.25	0.42	0.42	0.26	0.26	0.38	0.65

Based on these cross-conformance checking results, we grouped the sub-logs (i.e., batches) into *clusters* using k-means clustering. We chose this clustering algorithm because domain experts from Xerox indicated that a batch belongs to a cluster and cannot overlap to other clusters. Nevertheless, other non-overlapping clustering algorithms can be used instead.

Figure 11.8 shows the process mining (sub) workflow that implements the clustering step of this phase (i.e., step 3c in Figure 11.3).

In concrete, we used the rows of the cross-conformance matrix presented above as observations to perform a *k-means* clustering. We chose to use the rows because we want to cluster the batches themselves, not their process models (columns).

Figure 11.9 shows the result of the clustering of batches. For each batch, it indicates its membership probability related to each of the defined clusters. In this experiment, we used k = 3 clusters.

Based on these results, we can relate each batch to one of the three clusters. Finally, the composition of each batch is as follows:

- cluster 0: batches 3, 4, 18 and 23.
- cluster 1: batches 1, 2, 7, 11 and 58.
- cluster 2: batch 30.

The resulting clusters contain groups of "interesting" comparable (i.e., relatively-similar) batches. Now, in-depth comparative batch analysis can be performed within any cluster. Note that cluster 2 contains only one batch. This can be caused by the fact that batch 30 is very different from all other batches.

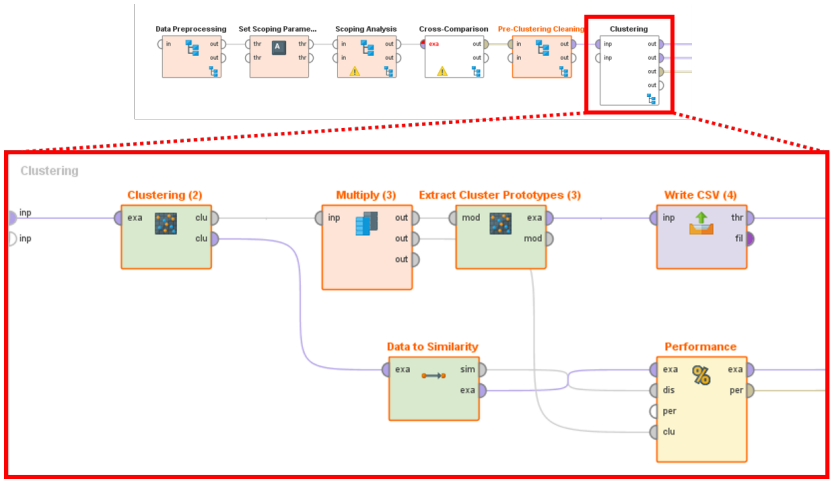


Figure 11.8: The RapidProM (sub) workflow used in this phase for step 3c (i.e., clustering) of the experimental design.

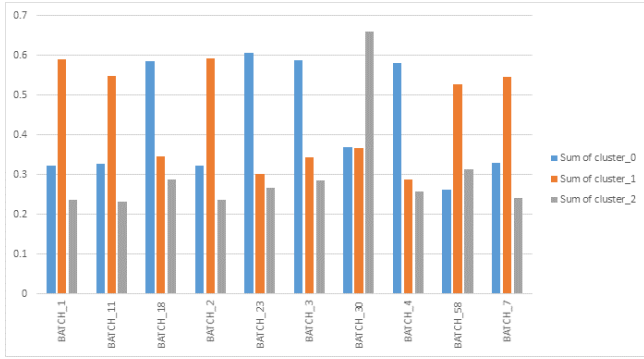


Figure 11.9: Results of the clustering step: The y-axis represents the cluster membership probabilities of batches. A batch will be related to the cluster with the maximal membership probability.

11.2.3 In-Depth Batch Comparison

Once clusters of comparable batches have been identified in the previous phase, we can proceed to phase 3 of the experiment: in-depth comparison of the

batches (i.e., step 4 in Figure 11.3) and interpretation and validation of the results (i.e., step 5 in Figure 11.3). In order to perform an in-depth comparison of the batches, we used the process comparison technique introduced in Chapter 5 for this purpose. This technique detects statistically significant differences between sub-logs in terms of control-flow and performance. The results of this technique identify those parts of the process where differences occur.

We first applied the *process comparator* tool to the four sub-logs of cluster 0 to get a list of pair-wise comparisons of sub-logs, sorted by similarity, i.e., percentage of control-flow differences. This pair-wise list generation function is explained in detail in Section 5.3. The results were:

1. batch 3 vs. batch 18 (38.04% control-flow differences)
2. batch 4 vs. batch 23 (42.03% control-flow differences)
3. batch 3 vs. batch 23 (72.16% control-flow differences)
4. batch 18 vs. batch 23 (73.40% control-flow differences)
5. batch 3 vs. batch 4 (78.43% control-flow differences)
6. batch 4 vs. batch 18 (78.64% control-flow differences)

This means that, in terms of control-flow, batches 4 and 18 are the most dissimilar pair within cluster 0, and batches 3 and 18 are the most similar.

In order to illustrate the in-depth comparison step, in the remainder of this section we will focus on analyzing the differences between batches 4 and 18 and between batches 3 and 18.

First, we checked for differences between batches 4 and 18 (the most dissimilar in cluster 0). Figure 11.10 shows an example of the control-flow differences found between batches 4 and 18. The dark-blue colored states are executed only in batch 18, and never in batch 4. These states are related to *optical character recognition* (OCR) in forms.

Moreover, an example of the performance differences found between batches 4 and 18 is shown in Figure 11.11. Note that the duration of the activity *Entry* is statistically significantly higher in batch 18 compared to in batch 4. This activity refers to manual entry of form content.

Then, we checked for differences between batches 3 and 18 (the most similar in cluster 0). Figure 11.10 shows a significant difference in the frequency of execution of the process fragment related to the transformation of data from XML to the X12 format, and the transmission and acknowledgment of that data.

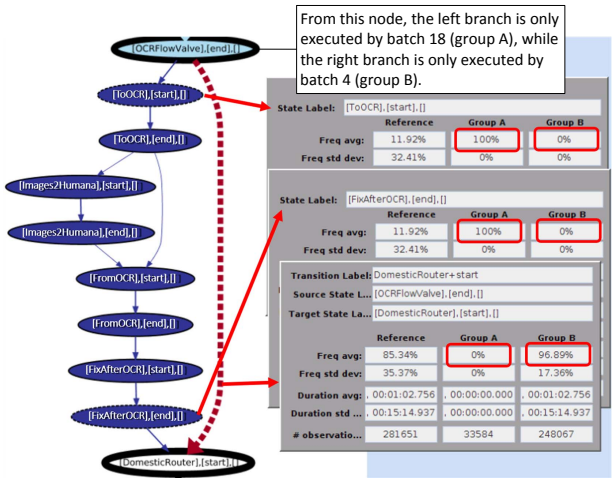


Figure 11.10: Example of control-flow differences between batch 18 (group A) and batch 4 (group B). The activities *ToOCR, Images2Humana, FromOCR, FixAfterOCR* are executed only in batch 18.

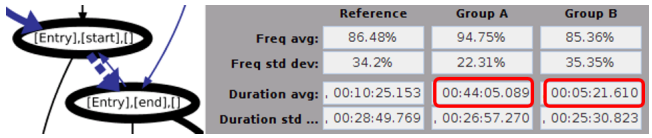


Figure 11.11: Example of performance differences between found batch 18 (group A) and batch 4 (group B). The average duration of the *Entry* activity is 44 mins for batch 18 and 5 mins for batch 4.

This fragment is almost always executed in batch 18. However, it is executed only in approximately 93% of the cases in batch 3. Similarly, the *Cleanup* activity is executed in only 5% of the cases in batch 18 against 12% of the cases in batch 3. From a performance point of view, we see that there is a significant difference in the average duration of cases until the execution of the *Cleanup* activity (22 days in batch 3 vs. 10 days in batch 18). However, it is important to also note that the standard deviation of the duration until *Cleanup* is very high relative to the average duration.

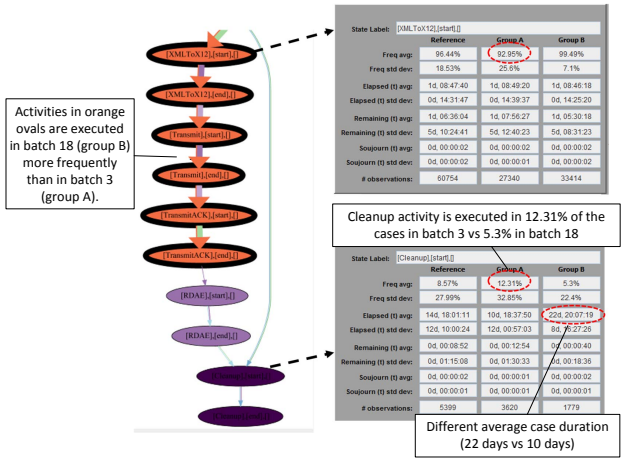


Figure 11.12: Example of differences found between batch 3 (group A) and batch 18 (group B).

11.3 Discussion

For the interpretation and validation of the results (i.e., step 5 in Figure 11.3), we presented the experiment results to a domain expert from Xerox, who confirmed our results by explaining the differences found in the batches.

According to the domain expert, the control-flow differences in Figure 11.10 are attributed to the fact that the two batches deal with different types of forms. Batch 18 deals with UB-04 forms (shown in Figure 11.1): a claim form used by hospitals, nursing facilities, in-patient, and other facility providers. These forms are filled by healthcare providers and they can contain handwriting (e.g., disease codes, diagnosis, etc.), so OCR is needed. In contrast, batch 4 deals with claim correspondence forms i.e., reply forms from the provider (shown in Figure 11.1). These forms are typically digital. Hence, there is no need for OCR.

The domain expert also stated that the performance difference shown in Figure 11.11 is attributed to the fact that the forms related to batch 4 (i.e., correspondence forms) are usually smaller than the forms related to batch 18 (i.e., UB-04 forms), and have little content to be entered manually. Hence, the average duration of *Entry* activity in batch 4 is lower. Although these differences between batch 18 and 4 are insightful, they are not very surprising once

explained. Similarly, the differences in duration in the manual entry of smaller vs. larger forms in terms of page and image count are to be expected as well.

The differences between batches 3 and 18 have also provided interesting actionable insights. Both the batches 3 and 18 correspond to a similar type of form (UB-04) and are expected to have very similar behavior. The remarkable differences in the frequencies in the process fragment are statistically significant and unexpected by the domain expert. Hence, they need to be investigated further. The observed differences in duration until the *Cleanup* activity could be explained by the fact that, in the analyzed process, a lot of (sub) batch processing is involved, and as such, cases sometimes need to wait for other cases in order to be processed.

During these experiments, we have obtained the results to answer the research questions posed in Section 11.1.3. Now, we will address them explicitly:

Regarding question 1 (i.e., what are the workflows followed by the different batches, what do they have in common, and where do they differ?) our answer is detailed in Section 11.2.3, where concrete differences are found in terms of control-flow and performance.

Regarding question 2 (i.e., can we come up with a standardized process model that covers all batches?), our answer is no. Batches related to different forms will require different activities (e.g., some batches require OCR and others do not, as shown in Figure 11.10).

Regarding question 3 (i.e., are there particular paths that influence process performance, and are these paths common across the batches?), our answer is yes for the first part and no for the second part. For example doing OCR requires more time than not doing it. However these paths are not commong across the batches. They occur only in batches related to certain forms.

11.4 Conclusion

In this chapter, we presented an application of the tools and techniques introduced in this thesis to perform a comparative analysis of business process outsourcing services provided by Xerox that relate to a form digitalization process. The process pertains to the data entry back-office operations of insurance claim forms. The organization is interested in analyzing the processes followed across different batches.

As there are 94 batches in total, it was not feasible to compare each pair of batches in detail. We used process mining workflows (see Chapter 4) to preprocess the data and cluster the batches based on their similarity. Then, we used

the process comparator tool (see Chapter 5) to perform an in-depth comparison of the batches within a cluster.

Finally, the results were interpreted and validated by domain experts, and the discovered insights and actions were delivered to the process owners.

Through the use of the tools and techniques introduced in this thesis, we could obtain very meaningful results, which have been confirmed by a domain expert and transformed into actionable insights, such as studying the root causes and contextual circumstances for the most important differences.

Part V

Closure

Chapter 12

Conclusions

This chapter concludes this thesis by first highlighting the contributions included in it. Then, it discusses the limitations of this thesis. Finally, it describes future work related to the topics included in this thesis.

12.1 Contributions Review

This section summarizes how each of the five research contributions presented in Section 1.4 were addressed in this thesis. These contributions are described as follows.

Process Cubes: *A technique to support the interactive and consistent exploration of process variants*.

An important feature in modern process mining is the ability to analyze and compare different variants (behaviors) of the process from different perspectives. In such way, organizations are able to see how processes can be improved by understanding differences between groups of cases, departments, etc.

In Chapter 3, we introduced *process cubes* as a way to organize, split and explore event data by using different data dimensions to split such event data into process variants. This way of splitting event data helps exposing differences between such variants, e.g., by using process variant comparison techniques.

We implemented this idea in our *PMC* tool, and we encourage the process mining community to use it. Such tool was used effectively in several case

studies presented in Part IV. In these case studies, the use of the *PMC* tool led to important insights that were confirmed by domain experts.

Process Variant Comparison: *A technique to compare process variants.*

Processes may change because of the influence of several factors, such as the period of the year, the geographical location of the process execution or the resource unit in charge. This leads to different process variants. The problem of comparing process variants is highly relevant, as it can show organizations which and why some of these variants perform better than others, or are executed differently.

In Chapter 5, we introduced a new technique based on transition systems that detects statistically-significant differences between process variants in terms of any measurement annotations (e.g., control-flow frequency, performance). This technique also shows the similarities and differences of the business rules (i.e., decision-making) between process variants, using only event logs as the input. Also, this technique can be combined with the previous contribution: process variants obtained from a process cube can be compared with this technique.

We implemented this technique in our *Process Comparator* tool. This tool was applied in several case studies presented in Part IV, in which we showed that the approach enables users to pinpoint important differences between process variants that previous approaches failed to provide. The insights obtained with this tool were confirmed and valued by domain experts.

Process Variant Detection: *A technique to detect relevant process variants in a general setting.*

One of many interesting types of analysis in process mining is to find (and then compare) process variants. However, in many real-life situations, event logs are made available without any additional domain knowledge about the data. This lack of domain knowledge makes finding process variants to be more difficult and cumbersome, as it requires extensive trial-and-error by the analyst.

In Chapter 6, we introduced an approach that is able to detect *relevant* process variants in any process perspective (in the form of event attributes) by splitting any other (combination of) event attributes. This technique can be combined with the previous contributions: identified process variants can be used by a process cube to split the event data into such variants, which then can be used in process mining workflows or compared using the process comparison tool.

This approach has been implemented in our *Process Variant Finder* tool. Us-

ing this tool, we were able to successfully identify points of process variability (i.e., variants) inside both artificial and real-life event logs and we were able to detect process variants without the use of domain knowledge, confirming such variability using process comparison techniques, as presented in the case studies included in Part IV. Therefore, our approach provides a viable solution to process variant detection, even when no domain knowledge is available.

Process Mining Workflows: *Support the execution of process mining workflows.*

Current scientific workflow systems are not tailored towards the analysis of processes based on models and logs. Tools like RapidMiner and KNIME can model analysis workflows but do not provide any process mining capabilities. The focus of these tools is mostly on traditional data mining and reporting capabilities that tend to use tabular data. On the other hand, process mining tools like ProM, Disco, Celonis, Everflow, QPR, MyInvenio, Minit, PM4PY, bupaR, etc. do not fully provide explicit workflow support.

In Chapter 4, we introduced process mining workflows by proposing several generic *process mining building blocks* that can be chained together to create such workflows. We identified five generic use cases for typical process mining analysis and provided conceptual and concrete workflows for them.

The whole approach is implemented as RapidProM, which is a ProM-based extension for RapidMiner. This tool was used in several case studies presented in Part IV, in which we exploited the advantages of process mining workflows for producing massive quantities of process-mining-based reports, and for combining process mining techniques with traditional data mining operations in the same workflow.

Benchmark Frameworks: *Develop replicable and sound benchmarks.*

Existing empirical evaluation frameworks in process mining have several drawbacks. Two of the main drawbacks are the replicability and soundness of the results. The first relates to the fact that experimental results are difficult to replicate using existing empirical evaluation frameworks in process mining, as they require multiple manual steps that are sensitive to parameterizations, including ad-hoc cleaning and preprocessing of the event data. The second relates to the fact that in most empirical evaluation frameworks in process mining, experiments are performed over manually-selected samples of event data and/or process models and then generalized to populations of processes that they do not represent.

In Part III we introduced two empirical evaluation frameworks for process

mining that overcame these limitations by combining process mining workflows and statistical analysis.

In Chapter 7, we presented a framework that allows researchers to benchmark discovery algorithms as well as to perform a sensitivity analysis to evaluate whether certain model or log characteristics have a significant effect on an algorithm's performance. It is independent from the discovered model's modeling notation by adopting a classification approach that uses the knowledge of the original (reference) model to be rediscovered. Additionally, the framework allows to generalize the evaluation results to a user specified model population.

In Chapter 8, we introduced an evaluation framework for benchmarking concept drift detection techniques. The framework allows researchers to benchmark different approaches as well as to perform a sensitivity analysis to evaluate whether certain process and drift characteristics have an impact on the accuracy drift point detection. It allows to generalize the evaluation results to a user specified model population, also taking into account several properties that characterize concept drift in processes.

Both frameworks have been validated by conducting extensive experiments that led to non-trivial insights on discovery and concept drift detection algorithms in the context of the populations of processes included in the experiments. Finally, the design of both framework as process mining workflows enable automating, sharing and extending these evaluation experiments.

12.2 Limitations

The previous section indicated how the different research contributions were addressed by the techniques and tools introduced in this thesis. In this Section, we will discuss the main limitations of these techniques and tools and propose ideas of how these limitations could be overcome in the future.

Regarding the *PMC tool* (introduced in Chapter 3), this tool has a set of functions that allows to perform all the described basic operations. However, more advanced functions such as direct integration to databases and custom hiding/merging cells of a cube are not implemented, as this tool is still a prototype. We invite interested readers to access the source code on GIT and make improvements to this tool.

With respect to the *process comparator* technique (introduced in Chapter 5), so far it can only compare annotations based on control-flow frequency and performance. Other types of annotations (e.g., cost, resource usage) can be easily added. However, the main limitation of this technique is that it still relies

on comparing one (process or group) versus another, projecting the differences onto a combined model. It would be even more useful to be able to compare many processes at the same time in a n-to-n fashion. However, if this n-to-n comparison would be naively projected onto a single combined model, it could lead to a visually overloaded result that would be difficult to read and analyze. We believe that alternative representations of comparison results must be investigated in order to achieve this.

Regarding the *process variant detection* tool (introduced in Chapter 6), it currently relies on the user indicating the independent variable (i.e., the variable in which we want to analyze variability). It would be much more useful if the user did not need to make such selection, specially in the case where no domain knowledge is available. Moreover, The tool only works in an *offline* setting (i.e., it does not work with event streams). A nice feature would be to adapt it to work with sliding windows so it can also work in an online setting.

12.3 Future Work

The techniques, tools and experiments developed and introduced in this thesis are also subject to improvements. This section describes our proposals for future work on them.

The *process mining workflow technique* (introduced in Chapter 4) has been successfully implemented as an extension of RapidMiner, but such as it is, it is only available to RapidMiner users. We would like to port this implementation to other platforms such as Knime or as stand-alone libraries so that these workflow can be used by a wider part of the research community.

The *process comparison technique* (introduced in Chapter 5) works very well comparing variants 1-to-1, but further work is needed to extend this comparison to n-to-n variants simultaneously.

Regarding both large-sacale experimental benchmarks introduced in Part III, we would like to continue expanding the list of model populations, modelling algorithms and notations used in the experiments in order to make even more robust assertions about the obtained results.

Last, but not least, we would like to integrate all of the tools and techniques introduced in Part II into a single integrated suite intended for the initial exploration of process data, identification and comparison of process variants, and their automated analysis.

Bibliography

[1] IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. *IEEE Std 1849-2016*, pages 1–50, Nov 2016. (Cited on page 70.)

[2] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment based precision checking. In *Business Process Management Workshops*, volume 132, pages 137–149. Springer Berlin Heidelberg, 2013. (Cited on pages 105, 108, and 110.)

[3] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Measuring precision of modeled behavior. *Information Systems and e-Business Management*, 13(1):37–67, 2015. (Cited on pages 105, 108, and 110.)

[4] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367, May 2017. (Cited on pages 225 and 233.)

[5] Rachmadita Andreswari and Mohammad Arif R. Olap cube processing of production planning real-life event log: A case study. In *Proceedings of the 2018 International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018)*, pages 148–153. Atlantis Press, 2019/03. (Cited on page 53.)

[6] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo.

Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Transactions on Knowledge and Data Engineering*, May 2018. (Cited on page 188.)

[7] Roger Barga and Dennis Gannon. Scientific versus business workflows. In Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 9–16. Springer Verlag, Berlin, 2007. (Cited on page 83.)

[8] Adam Barker and Jano van Hemert. Scientific workflow: A survey and research directions. In *Parallel Processing and Applied Mathematics*, pages 746–753. Springer Berlin Heidelberg, 2008. (Cited on pages 189 and 226.)

[9] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Koetter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. Knime: The Konstanz information miner. In Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker, editors, *Data Analysis, Machine Learning and Applications*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 319–326. Springer Berlin Heidelberg, 2008. (Cited on pages 81 and 84.)

[10] Alessandro Berti, Sebastiaan J van Zelst, and Wil van der Aalst. Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. page 13–16, 2019. (Cited on page 84.)

[11] Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. Scientific workflows for process mining: building blocks, scenarios, and implementation. *International Journal on Software Tools for Technology Transfer*, 18(6):607–628, Nov 2016. (Cited on pages 26 and 368.)

[12] Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. A visual approach to spot statistically-significant differences in event logs based on process metrics. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering*, pages 151–166, Cham, 2016. Springer International Publishing. (Cited on pages 26, 123, and 368.)

[13] Alfredo Bolt, Massimiliano de Leoni, and Wil M.P. van der Aalst. Process variant comparison: Using event logs to detect differences in behavior

and business rules. *Information Systems*, 74:53 – 66, 2018. Information Systems Engineering: selected papers from CAiSE 2016. (Cited on pages 26, 123, and 368.)

[14] Alfredo Bolt, Massimiliano De Leoni, Wil M.P. van der Aalst, and Pierre Gorissen. Exploiting process cubes, analytic workflows and process mining for business process reporting: A case study in education. In P. Ceravolo and S. Rinderle-Ma, editors, *Data-Driven Process Discovery and Analysis (SIMPDA 2015), December 9-11, 2015, Vienna, Austria*, CEUR Workshop Proceedings, pages 33–47. CEUR-WS.org, 2015. 5th International Symposium on Data-Driven Process Discovery and Analysis, SIMPDA 2015 ; Conference date: 09-12-2015 Through 11-12-2015. (Cited on page 369.)

[15] Alfredo Bolt and Marcos Sepúlveda. Process remaining time prediction using query catalogs. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, pages 54–65, Cham, 2014. Springer International Publishing. (Cited on page 370.)

[16] Alfredo Bolt and Wil M. P. van der Aalst. Multidimensional process mining using process cubes. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 102–116, Cham, 2015. Springer International Publishing. (Cited on pages 25 and 368.)

[17] Alfredo Bolt, Wil M. P. van der Aalst, and Massimiliano de Leoni. Finding process variants in event logs. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 45–52, Cham, 2017. Springer International Publishing. (Cited on pages 26 and 368.)

[18] A.J. Bolt Iriondo, M. de Leoni, W.M.P. van der Aalst, and P. Gorissen. Business process reporting using process mining, analytic workflows and process cubes: A case study in education. In C. Paolo and R.-M. Stefanie, editors, *Data-Driven Process Discovery and Analysis*, Lecture Notes in Business Information Processing, pages 28–53, Germany, 2017. Springer. 5th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2015, SIMPDA 2015 ; Conference date: 09-12-2015 Through 11-12-2015. (Cited on page 369.)

[19] Diana Borrego and Irene Barba. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Systems with Applications*, 41(11):5340 – 5352, 2014. (Cited on page 204.)

[20] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indrè Žliobaite, and Mykola Pechenizkiy. Dealing with concept drifts in process mining. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):154–171, Jan 2014. (Cited on page 225.)

[21] R. P. Jagadeesh Chandra Bose and Wil M.P. van der Aalst. Context Aware Trace Clustering: Towards Improving Process Mining Results. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412, 2009. (Cited on page 159.)

[22] Cynthia Brewer. *Designing Better Maps: A Guide for Gis Users*. Environmental Systems Research, 2004. (Cited on pages 132 and 134.)

[23] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), coselog project, municipality 1. `10.4121/uuid:c45dcbe9-557b-43ca-b6d0-10561e13dcb5`, 2014. (Cited on page 5.)

[24] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), coselog project, municipality 2. `10.4121/uuid:34b4f6f4-dbe0-4857-bf75-5b9e1138eb87`, 2014. (Cited on page 5.)

[25] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), coselog project, municipality 3. `10.4121/uuid:a8ed945d-2ad8-480e-8348-cf7f06c933b3`, 2014. (Cited on page 5.)

[26] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), coselog project, municipality 4. `10.4121/uuid:e8c3a53d-5301-4afb-9bcd-38e74171ca32`, 2014. (Cited on page 5.)

[27] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), coselog project, municipality 5. `10.4121/uuid:c399c768-d995-4086-adda-c0bc72ad02bc`, 2014. (Cited on page 5.)

[28] Joos C. A. M. Buijs and Hajo A. Reijers. Comparing business process variants using models and event logs. In Ilia Bider, Khaled Gaaloul, John Krogstie, Selmin Nurcan, Henderik A. Proper, Rainer Schmidt, and Pnina Soffer, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 154–168, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. (Cited on page 320.)

[29] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In Robert Meersman, Hervé Panetto, Tharam Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *On the Move to Meaningful Internet Systems (OTM)*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2012. (Cited on page 104.)

[30] Andrea Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016)*, volume 1789 of *CEUR Workshop Proceedings*, pages 1–6. CEUR-WS.org, 2017. (Cited on pages 189 and 195.)

[31] Andrea Burattin, Fabrizio M. Maggi, and Alessandro Sperduti. Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications*, 65:194 – 211, 2016. (Cited on page 204.)

[32] Andrea Burattin and Alessandro Sperduti. PLG: a framework for the generation of business process models and their execution logs. In Michael zur Muehlen and Jianwen Su, editors, *Business Process Management Workshops*, pages 214–219, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on pages 94 and 195.)

[33] Josep Carmona and Ricard Gavaldà. Online techniques for dealing with concept drift in process mining. In Jaakko Hollmén, Frank Klawonn, and Allan Tucker, editors, *Advances in Intelligent Data Analysis XI*, pages 90–102, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on page 225.)

[34] Malu Castellanos, Ana Karla Alves de Medeiros, Jan Mendling, Barbara Weber, and Anton J. M. M. Weijers. Business process intelligence. In Jorge Cardoso and Wil M. P. van der Aalst, editors, *Handbook of Research on Business Process Modeling*, chapter 21, pages 456–480. IGI Global, Hershey, PA, USA, 2009. (Cited on page 290.)

[35] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, March 1997. (Cited on page 51.)

[36] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and S Yu Philip. Graph olap: a multi-dimensional framework for graph data analysis. *Knowledge and Information Systems*, 21(1):41–63, 2009. (Cited on page 51.)

[37] Claudio Di Ciccio and Massimo Mecella. On the discovery of declarative control flows for artful processes. *ACM Transactinons on Managemet Information Systems*, 5(4):24:1–24:37, January 2015. (Cited on pages 203 and 205.)

[38] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988. (Cited on pages 131, 194, and 229.)

[39] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. BPMN miner: Automated discovery of BPMN process models with hierarchical structure. *Information Systems*, 56:284 – 303, 2016. (Cited on page 38.)

[40] Carsten Cordes, Thomas Vogelgesang, and Hans-Jürgen Appelrath. A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In *Business Process Management Workshops*, volume 202 of *Lecture Notes in Business Information Processing*, pages 383–394. Springer International Publishing, 2015. (Cited on page 122.)

[41] Thomas H. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, Boston, Massachusetts, 1993. (Cited on page 3.)

[42] Massimiliano de Leoni, Fabrizio M. Maggi, and Wil M.P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems*, 47:258 – 277, 2015. (Cited on page 204.)

[43] Massimiliano. De Leoni and Felix Mannhardt. Road traffic fine management process, 2015. (Cited on page 42.)

[44] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments. In *28th ACM symposium on Applied Computing (SAC'13)*, pages 1454–1461. ACM, 2013. (Cited on page 135.)

[45] Massimiliano de Leoni, Wil M.P. van der Aalst, and Marcus Dees. A General Process Mining Framework for Correlating, Predicting and Clustering Dynamic Behavior based on Event Logs. *Information Systems*, 56:235 – 257, 2016. (Cited on pages 34, 35, 105, 160, and 168.)

[46] Jochen De Weerdt. *Business process discovery: new techniques and applications*. PhD thesis, KU Leuven, 2012. (Cited on page 188.)

[47] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A Multi-dimensional Quality Assessment of State-of-the-art Process Discovery Algorithms Using Real-life Event Logs. *Information Systems*, 37(7):654–676, November 2012. (Cited on pages 185 and 188.)

[48] Angela Dean, Daniel Voss, and Danel Draguljić. *Design and Analysis of Experiments*. Springer-Verlag New York, 1999. (Cited on pages 190, 191, 206, 207, and 238.)

[49] Claudia Diamantini, Domenico Potena, and Emanuele Storti. Mining usage patterns from a repository of scientific workflows. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 152–157, New York, NY, USA, 2012. ACM. (Cited on page 84.)

[50] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013. (Cited on pages 83 and 145.)

[51] Iulia. Efremova, Alejandro Montes Garcia, Alfredo. Bolt, and Toon G.K. Calders. Who are my ancestors? : retrieving family relationships from historical texts. In P. Braslavski , I. Markov, P. Pardalos, Y. Volkovich, D.I. Ignatov , S. Koltsov, and O. Koltsova, editors, *Information Retrieval*, Communications in Computer and Information Science, pages 121–129, Germany, 2015. Springer. (Cited on page 370.)

[52] Dirk Fahland and Wil M.P. van der Aalst. Model repair — aligning process models to reality. *Information Systems*, 47:220 – 243, 2015. (Cited on page 105.)

[53] Mohammadreza Fani Sani, Wil van der Aalst, Alfredo Bolt, and Javier García-Algarra. Subgroup discovery in process mining. In Witold Abramowicz, editor, *Business Information Systems*, pages 237–252, Cham, 2017. Springer International Publishing. (Cited on page 369.)

[54] Rebecca Ferguson. Learning analytics: drivers, developments and challenges. *International Journal of Technology Enhanced Learning*, 4(5-6):304–317, 2012. (Cited on pages 292 and 310.)

[55] Ronald A. Fisher. *The Design of Experiments*, volume 12. Oliver and Boyd Edinburgh, 1960. (Cited on pages 190 and 226.)

[56] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014. (Cited on pages 160 and 224.)

[57] Daniel Garijo, Pinar Alper, Khalid Belhajjame, Óscar Corcho, Yolanda Gil, and Carole A. Goble. Common motifs in scientific workflows: An empirical analysis. *Future Generation Comp. Syst.*, 36:338–351, 2014. (Cited on page 84.)

[58] Christian W. Günther. *Process Mining in Flexible Environments*. PhD thesis, Technische Universiteit Eindhoven, 2009. (Cited on pages 106 and 200.)

[59] Carole A. Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li, and David De Roure. myExperiment: A repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(suppl 2):W677–W682, 2010. (Cited on page 83.)

[60] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010. (Cited on page 83.)

[61] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10(Jun):1305–1340, 2009. (Cited on page 189.)

[62] Petrus J. B. Gorissen. *Facilitating the use of recorded lectures: analysing students' interactions to understand their navigational needs*. PhD thesis, Eindhoven University of Technology, 2013. (Cited on pages 291 and 310.)

[63] Daniela Grigori, Fabio Casati, Malu Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business process intelligence.

Computers in Industry, 53(3):321 – 343, 2004. Process / Workflow Mining. (Cited on pages 290 and 291.)

[64] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management: 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007. Proceedings*, pages 328–343, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. (Cited on pages 16, 69, 104, 292, and 296.)

[65] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, Cambridge, MA, USA, 2001. (Cited on page 83.)

[66] Markus Hofmann and Ralf Klinkenberg. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman and Hall/CRC, 2013. (Cited on pages 81, 84, and 100.)

[67] Bart Hompes, Joos C. A. M. Buijs, Wil M. P. van der Aalst, Prabhakar Dixit, and Johannes Buurman. Detecting change in processes using comparative trace clustering. In *5th International Symposium on data-driven process discovery and analysis (SIMPDA)*, pages 95–108, 2015. (Cited on page 160.)

[68] Bart F. A. Hompes, Joos C. A. M. Buijs, and Wil M. P. van der Aalst. A generic framework for context-aware process performance analysis. In *Proceedings of CoopIS 2016*, pages 300–317. Springer International Publishing, 2016. (Cited on page 160.)

[69] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006. (Cited on page 168.)

[70] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. CTREE: Conditional inference trees. *Cran. R_project*, 2015. (Cited on page 171.)

[71] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole A. Goble, Matthew R. Pocock, Peter Li, and Tom Oinn. Taverna: A tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006. (Cited on pages 81 and 83.)

[72] IEEE Task Force on Process Mining. Process Mining Case Studies. http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies, 2013. (Cited on pages 9 and 21.)

[73] Sergey Ivanov, Anna Kalenkova, and Wil M. P. van der Aalst. BPMNDiffViz: A tool for BPMN models comparison. In *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015.*, pages 35–39, 2015. (Cited on page 122.)

[74] Stefan Jablonski, Maximilian Röglinger, Stefan Schönig, and Katrin Maria Wyrtki. Multi-perspective clustering of process execution traces. *Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling*, 14(2):1–22, 2019. (Cited on page 160.)

[75] R.P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In *Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 159–175. Springer Berlin Heidelberg, 2009. (Cited on page 123.)

[76] Gert Janssenswillen, Benoît Depaire, and Toon Jouck. Calculating the number of unique paths in a block-structured process model. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016*, pages 138–152. CEUR Workshop Proceedings, 2016. (Cited on page 206.)

[77] Gert Janssenswillen, Benoît Depaire, Marijke Swennen, Mieke Jans, and Koen Vanhoof. bupar: Enabling reproducible business process analysis. *Knowledge-Based Systems*, 163:927 – 930, 2019. (Cited on page 84.)

[78] Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: a Classification Perspective*. Cambridge University Press, 2011. (Cited on page 189.)

[79] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002. (Cited on page 193.)

[80] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013. (Cited on page 36.)

[81] Micheline Kamber Jiawei Han, Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2011. (Cited on page 47.)

[82] Tao Jin, Jianmin Wang, and Lijie Wen. Generating benchmarks by random stepwise refinement of Petri nets. In *Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency, and of the 10th International Conference on Application of Concurrency to System Design*, volume 827 of *CEUR Workshop Proceedings*, pages 403–417. CEUR-WS.org, 2010. (Cited on page 195.)

[83] Tao Jin, Jianmin Wang, and Lijie Wen. Efficiently querying business process models with beehivez. In *Proceedings of the Demo Track of the Nineth Conference on Business Process Management 2011*, volume 820 of *CEUR Workshop Proceedings*, pages 1–6. CEUR-WS.org, 2011. (Cited on page 195.)

[84] Toon Jouck. *Empirically Evaluating Process Mining Algorithms: Towards Closing the Methodological Gap*. PhD thesis, Hasselt University, 2018. (Cited on page 185.)

[85] Toon Jouck, Alfredo Bolt, Benoît Depaire, Massimiliano de Leoni, and Wil MP van der Aalst. An integrated framework for process discovery algorithm evaluation. Technical report, https://arxiv.org/abs/1806.07222, 2018. (Cited on pages 26, 185, and 370.)

[86] Toon Jouck and Benoît Depaire. Ptandloggenerator: A generator for artificial event data. In *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.*, pages 23–27, 2016. (Cited on pages 94, 102, and 104.)

[87] Toon Jouck and Benoît Depaire. Simulating Process Trees Using Discrete-Event Simulation. Technical Report, Hasselt University, February 2017. (Cited on page 189.)

[88] Toon Jouck and Benoît Depaire. Generating Artificial Data for Empirical Analysis of Control-flow Discovery Algorithms: A Process Tree and Log Generator. *Business & Information Systems Engineering, vol 3/2018*, pages 1–18, March 2018. (Cited on pages 189, 195, 197, and 229.)

[89] Anna Kalenkova, Massimiliano de Leoni, and Wil M. P. van der Aalst. Discovering, analyzing and enhancing BPMN models using prom. In *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014)*, volume 1295 of *CEUR Workshop Proceedings*, pages 36–40. CEUR-WS.org, 2014. (Cited on page 202.)

[90] Valeriia Kataeva and Anna A. Kalenkova. Applying graph grammars for the generation of process models and their logs. In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, number 8, 2014. (Cited on page 195.)

[91] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual analytics: Definition, process, and challenges. In Andreas Kerren, JohnT. Stasko, Jean-Daniel Fekete, and Chris North, editors, *Information Visualization*, volume 4950 of *Lecture Notes in Computer Science*, pages 154–175. Springer Berlin Heidelberg, 2008. (Cited on page 96.)

[92] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011. (Cited on pages 47 and 51.)

[93] Roger E. Kirk. *Experimental Design*. John Wiley and Sons, 1982. (Cited on pages 189, 190, and 226.)

[94] Christopher Klinkmüller and Ingo Weber. Analyzing control flow information to improve the effectiveness of process model matching techniques. *Decision Support Systems*, 100:6–14, 2017. Smart Business Process Management. (Cited on page 122.)

[95] Christopher Klinkmüller and Ingo Weber. Every apprentice needs a master: Feedback-based effectiveness improvements for process model matching. *Information Systems*, 95:101612, 2021. (Cited on page 122.)

[96] Janez Kranjc, Vid Podpečan, and Nada Lavrač. Clowdflows: A cloud based scientific workflow platform. In PeterA. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 816–819. Springer Berlin Heidelberg, 2012. (Cited on page 83.)

[97] Simone Kriglstein, Günter Wallner, and Stefanie Rinderle-Ma. A visualization approach for difference analysis of process models and instance traffic. In *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 219–226. Springer Berlin Heidelberg, 2013. (Cited on page 122.)

[98] Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards Understanding Process Modeling — the Case of the BPM Academic Initiative. In *International Workshop on Business Process Modeling Notation*, pages 44–58. Springer, 2011. (Cited on pages 206 and 238.)

[99] Marcello La Rosa, Marlon Dumas, Reina Uba, and Remco Dijkman. Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol.*, 22(2):11:1–11:42, March 2013. (Cited on page 122.)

[100] Marcello La Rosa, Hajo A. Reijers, Wil M. P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. Apromore: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029–7040, 2011. (Cited on pages 84 and 94.)

[101] Geetika T. Lakshmanan, Szabolcs Rozsnyai, and Fei Wang. Investigating clinical care pathways correlated with outcomes. In *Business Process Management (BPM 2013)*, volume 8094 of *Lecture Notes in Computer Science*, pages 323–338. Springer Berlin Heidelberg, 2013. (Cited on page 123.)

[102] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In José-Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, pages 311–329, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. (Cited on pages 37, 69, 74, 76, 86, 94, 104, 110, 112, 203, 205, and 320.)

[103] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Exploring processes and deviations. In Fabiana Fournier and Jan Mendling, editors, *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pages 304–316, Cham, 2015. Springer International Publishing. (Cited on pages 106 and 296.)

[104] Frank Leymann and Dieter Roller. *Production workflow - concepts and techniques*. Prentice Hall, 2000. (Cited on page 83.)

[105] Xiaolei Li and Jiawei Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 447–458. VLDB Endowment, 2007. (Cited on page 51.)

[106] Richard Littauer, Karthik Ram, Bertram Ludäscher, William Michener, and Rebecca Koskela. Trends in use of scientific workflows: Insights from a public repository and recommendations for best practice. *IJDC*, 7(2):92–100, 2012. (Cited on page 84.)

[107] Mo Liu, Elke Rundensteiner, Kara Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. E-cube: Multi-dimensional event sequence processing using concept and pattern hierarchies. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 1097–1100. IEEE, 2010. (Cited on page 51.)

[108] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B. Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006. (Cited on pages 81 and 83.)

[109] Daniela Luengo and Marcos Sepúlveda. Applying clustering in process mining to find different versions of a business process that changes over time. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, pages 153–158, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on page 225.)

[110] Linh Thao Ly, Conrad Indiono, Jürgen Mangler, and Stefanie Rinderle-Ma. *Data Transformation and Semantic Log Purging for Process Mining*, pages 238–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. (Cited on page 292.)

[111] A. Maaradji, M. Dumas, M. L. Rosa, and A. Ostovar. Detecting sudden and gradual drifts in business processes from execution traces. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2140–2154, Oct 2017. (Cited on pages 225, 235, 237, 239, 243, 246, 248, 251, 254, and 256.)

[112] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. Fast and accurate business process drift detection. In *Business Process Management: 13th International BPM Conference*, pages 406–422. Springer International Publishing, 2015. (Cited on pages 160 and 225.)

[113] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 03 1947. (Cited on page 131.)

[114] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Balanced Multi-perspective Checking of Process Conformance. *Computing*, 98(4):407–437, 2016. (Cited on page 204.)

[115] Ronny S. Mans, Wil M. P. van der Aalst, and H. M. W. Verbeek. Supporting process mining workflows with RapidProM. In Lior Limonad and Barbara Weber, editors, *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM)*, volume 1295 of *CEUR Workshop Proceedings*, pages 56–60. CEUR-WS.org, 2014. (Cited on pages 84 and 100.)

[116] Jevgeni. Martjushev, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. Change point detection and dealing with gradual and multi-order dynamics in process mining. In Raimundas Matulevičius and Marlon Dumas, editors, *Perspectives in Business Informatics Research*, pages 161–178, Cham, 2015. Springer International Publishing. (Cited on pages 160, 225, 235, 237, 239, 243, 246, 248, 251, 254, and 256.)

[117] Jose-Norberto Mazón, Jens Lechtenbörger, and Juan Trujillo. A survey on summarizability issues in multidimensional modeling. *Data & Knowledge Engineering*, 68(12):1452–1469, 2009. (Cited on page 51.)

[118] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997. (Cited on page 83.)

[119] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, Fabrizio M. Maggi, and Suriadi Suriadi. Mining business process deviance: A quest for accuracy. In *On the Move to Meaningful Internet Systems (OTM 2014)*, volume 8841 of *Lecture Notes in Computer Science*, pages 436–445. Springer Berlin Heidelberg, 2014. (Cited on page 123.)

[120] Tapio Niemi, Marko Niinimäki, Peter Thanisch, and Jyrki Nummenmaa. Detecting summarizability in olap. *Data & Knowledge Engineering*, 89(Supplement C):1–20, 2014. (Cited on page 51.)

[121] Irene Ntoutsi, Alexandros Kalousis, and Yannis Theodoridis. A general framework for estimating similarity of datasets and decision trees: exploring semantic similarity of decision trees. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 810–821. SIAM, 2008. (Cited on page 123.)

[122] Petra Perner. How to compare and interpret two learnt decision trees from the same domain? In *Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 318–322. IEEE, 2013. (Cited on page 123.)

[123] Michael E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance (1985)*. Free Press, 2008. (Cited on page 4.)

[124] John R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. (Cited on page 135.)

[125] John R. Quinlan. *C4. 5: Programs for Machine Learning*. Elsevier, 2014. (Cited on pages 137 and 168.)

[126] Joel Ribeiro and Josep Carmona. A method for assessing parameter impact on control-flow discovery algorithms. In *Transactions on Petri Nets and Other Models of Concurrency XI*, pages 181–202. Springer Berlin Heidelberg, 2016. (Cited on page 188.)

[127] Joel Ribeiro, Josep Carmona, Mustafa Misir, and Michele Sebag. A recommender system for process discovery. In *Business Process Management*, pages 67–83. Springer International Publishing, 2014. (Cited on page 188.)

[128] Joel T. S. Ribeiro and Anton J. M. M. Weijters. Event cube: Another perspective on business processes. In Robert Meersman, Tharam Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng-Chin Ooi, Ernesto Damiani, Douglas C. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011: Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part I*, pages 274–283, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on page 52.)

[129] C. Romero and S. Ventura. Educational data mining: A review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics (Part C: Applications and Reviews)*, 40(6):601–618, Nov 2010. (Cited on pages 291 and 310.)

[130] Anne Rozinat, A. K. Alves de Medeiros, Christian W. Günther, Anton J. M. M. Weijters, and Wil M. P. van der Aalst. Towards an Evaluation Framework for Process Mining Algorithms. *BPM Center Report*, 07-06, 2007. (Cited on page 188.)

[131] Anne Rozinat and Wil M. P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008. (Cited on page 204.)

[132] Nick Russell, Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, and Nataliya Mulyar. Workflow Controlflow patterns: A Revised View. *BPM Center Report*, 06-22, 2006. (Cited on page 195.)

[133] August-Wilhelm Scheer and Markus Nüttgens. Aris architecture and reference models for business process management. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer Berlin Heidelberg, 2000. (Cited on page 94.)

[134] Jeffrey C. Schlimmer and Richard H. Granger, Jr. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, pages 502–507. AAAI Press, 1986. (Cited on page 225.)

[135] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. Detecting Concept Drift in Processes Using Graph Metrics on Process Graphs. In *Proceedings of the 9th Conference on Subject-oriented Business Process Management*, pages 6:1–6:10. ACM, 2017. (Cited on page 160.)

[136] Sidney Siegel and N. J. Castellan Jr. *Nonparametric statistics for the behavioral sciences*. Mcgraw-Hill, New York, 2 edition, 1988. (Cited on pages 207, 208, and 238.)

[137] G. Siemens. Learning analytics: The emergence of a discipline. *American Behavioral Scientist*, 57(10):1380–1400, 2013. (Cited on pages 291 and 310.)

[138] George Siemens and Ryan S. J. d. Baker. Learning analytics and educational data mining: Towards communication and collaboration. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, LAK '12, pages 252–254, New York, NY, USA, 2012. ACM. (Cited on pages 292 and 310.)

[139] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops: BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, pages 109–120, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on page 159.)

[140] Minseok Song and Wil M. P. van der Aalst. Supporting process mining by showing events at a glance. In *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145, 2007. (Cited on pages 105 and 296.)

[141] Mirko Sonntag, Dimka Karastoyanova, and Ewa Deelman. Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows. In *Sixth International Conference on e-Science, e-Science 2010, 7-10 December 2010, Brisbane, QLD, Australia*, pages 206–213, 2010. (Cited on page 83.)

[142] Bernhard Steffen, Tiziana Margaria, Ralf Nagel, Sven Joerges, and Christian Kubczak. Model-driven development with the jABC. In Eyal Bin, Avi Ziv, and Shmuel Ur, editors, *Hardware and Software, Verification and Testing*, volume 4383 of *Lecture Notes in Computer Science*, pages 92–108. Springer Berlin Heidelberg, 2007. (Cited on page 83.)

[143] Helmut Strasser and Christian Weber. The asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8:220–250, 1999. (Cited on page 168.)

[144] Jo Swinnen, Benoît Depaire, Mieke J. Jans, and Koen Vanhoof. A process deviation analysis – a case study. In *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 87–98. Springer Berlin Heidelberg, 2012. (Cited on page 123.)

[145] Alifah Syamsiyah, Alfredo Bolt, Long Cheng, Bart F. A. Hompes, R. P. Jagadeesh Chandra Bose, Boudewijn F. van Dongen, and Wil M. P.

van der Aalst. Business process comparison: A methodology and case study. In Witold Abramowicz, editor, *Business Information Systems*, pages 253–267, Cham, 2017. Springer International Publishing. (Cited on page 369.)

[146] Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil M. P. van der Aalst. The imprecisions of precision measures in process mining. *Information Processing Letters*, 135:1 – 8, 2018. (Cited on page 188.)

[147] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer Verlag, Berlin, 2007. (Cited on page 83.)

[148] Farbod Taymouri, Marcello La Rosa, and Josep Carmona. Business process variant analysis based on mutual fingerprints of event logs. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering*, pages 299–318, Cham, 2020. Springer International Publishing. (Cited on page 123.)

[149] Nikola Trcka, Mykola Pechenizkiy, and Wil M. P. van der Aalst. Process mining from educational data. In *Handbook of educational data mining*, chapter 9, pages 123–142. CRC Press, London, 2010. (Cited on pages 292 and 296.)

[150] Kenneth J. Turner and Paul S. Lambert. Workflows for quantitative data analysis in the social sciences. *International Journal on Software Tools for Technology Transfer*, pages 1–18, 2014. (Cited on page 84.)

[151] Nick van Beest, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Log delta analysis: Interpretable differencing of business process event logs. In *Proceedings of the 13th International Conference on Business Process Management (BPM'15)*, pages 386–405, 2015. (Cited on pages 122 and 123.)

[152] Wil van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian Günther, Antonella Guzzo, Paul Harmon,

Arthur ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari-Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Wynn. *Process Mining Manifesto*, pages 169–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. (Cited on pages 14, 21, 22, 83, and 223.)

[153] Wil M. P. van der Aalst. A decade of business process management conferences: Personal reflections on a developing discipline. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2012. (Cited on page 86.)

[154] Wil M. P. van der Aalst. Decomposing process mining problems using passages. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 72–91. Springer Berlin Heidelberg, 2012. (Cited on pages 92 and 98.)

[155] Wil M. P. van der Aalst. Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining. In Minseok Song, Moe Thandar Wynn, and Jianxun Liu, editors, *Asia Pacific Business Process Management: First Asia Pacific Conference, AP-BPM 2013, Beijing, China, August 29-30, 2013. Selected Papers*, pages 1–22, Cham, 2013. Springer International Publishing. (Cited on pages 51, 52, 53, and 79.)

[156] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. (Cited on pages vii, 4, 9, 15, 19, 36, 39, 93, 95, 96, 108, 126, 145, 185, 188, 193, 198, and 218.)

[157] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. (Cited on pages 69, 106, 108, and 296.)

[158] Wil M. P. van der Aalst, Joos Buijs, and Boudewijn van Dongen. Towards improving the representational bias of process mining. In Karl Aberer, Ernesto Damiani, and Tharam Dillon, editors, *Data-Driven Process Discovery and Analysis: First International Symposium, SIMPDA 2011, Campione d'Italia, Italy, June 29 – July 1, 2011, Revised Selected Papers*, pages 39–54, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on pages 37 and 229.)

[159] Wil M. P. van der Aalst, Alexander Dreiling, Florian Gottschalk, Michael Rosemann, and Monique H. Jansen-Vullers. Configurable process models as a basis for reference modeling. In Christoph J. Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 512–518. Springer Berlin Heidelberg, 2006. (Cited on page 96.)

[160] Wil M. P. van der Aalst and Schahram Dustdar. Process mining put into context. *IEEE Internet Computing*, 16(1):82–86, 2012. (Cited on page 119.)

[161] Wil M. P. van der Aalst, Shengnan Guo, and Pierre Gorissen. Comparative process mining in education: An approach based on process cubes. In Paolo Ceravolo, Rafael Accorsi, and Philippe Cudre-Mauroux, editors, *International Symposium of Data-Driven Process Discovery and Analysis: SIMPDA 2013*, pages 110–134. Springer Berlin Heidelberg, 2015. (Cited on pages 53 and 310.)

[162] Wil M. P. van der Aalst, Vladimir Rubin, Henricus M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87, Nov 2008. (Cited on pages 41, 42, 104, and 132.)

[163] Wil M. P. van der Aalst, Helen Schonenberg, and Minseok Song. Time prediction based on process mining. *Information Systems*, 36(2):450 – 475, 2011. Special Issue: Semantic Integration of Data, Multimedia, and Services. (Cited on page 128.)

[164] Wil M. P. van der Aalst and Minseok Song. Mining social networks: Uncovering interaction patterns in business processes. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin Heidelberg, 2004. (Cited on page 104.)

[165] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. (Cited on page 83.)

[166] Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve, and Moe T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011. (Cited on page 94.)

[167] Wil M. P. Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. (Cited on pages 37, 86, 94, 104, 190, 203, 205, and 320.)

[168] Wil M.P. van der Aalst. Relating process models and event logs-21 conformance propositions. In *ATAED@ Petri Nets/ACSD*, pages 56–74, 2018. (Cited on page 188.)

[169] Wil MP van der Aalst, Alfredo Bolt, and Sebastiaan J van Zelst. Rapidprom: mine your processes and not just your data. *arXiv preprint arXiv:1703.03740*, 2017. (Cited on pages 26, 100, and 370.)

[170] Jan M. E. M. van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In Kees M. van Hee and Rüdiger Valk, editors, *Applications and Theory of Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer Berlin Heidelberg, 2008. (Cited on pages 104, 203, 205, 219, 296, and 320.)

[171] Boudewijn F. van Dongen. Real-life event logs - hospital log. `10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`, 2011. (Cited on page 15.)

[172] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, pages 444–454, 2005. (Cited on pages 8, 21, 84, 100, 143, and 171.)

[173] Yoran P. J. M. van Oirschot. Using Trace Clustering for Configurable Process Discovery Explained by Event Log Data. Master's thesis, Eindhoven

University of Technology, Eindhoven, the Netherlands, 2014. (Cited on page 159.)

[174] Sebastiaan J. van Zelst, Alfredo Bolt , and Boudewijn F. van Dongen. Tuning alignment computation: an experimental evaluation. CEUR Workshop Proceedings, pages 6–20, 2017. Workshop on Algorithms and theories for the analysis of event data (ATAED2017), 26-27 June 2017, Zaragoza, Spain. (Cited on page 369.)

[175] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8:269–284, 2019. (Cited on page 368.)

[176] Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. *Computing alignments of event data and process models*, pages 1–26. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Germany, January 2018. (Cited on page 369.)

[177] Seppe K. L. M. vanden Broucke and Jochen De Weerdt. Fodina: A robust and flexible heuristic process discovery technique. *Decision Support Systems*, 100:109–118, August 2017. (Cited on page 188.)

[178] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. Determining process model precision and generalization with weighted artificial negative events. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1877–1889, 2014. (Cited on page 189.)

[179] Seppe K. L. M. vanden Broucke, Cédric Delvaux, João Freitas, Taisiia Rogova, Jan Vanthienen, and Bart Baesens. Uncovering the relationship between event log characteristics and process discovery techniques. In *Business Process Management Workshops*, pages 41–53, Cham, 2014. Springer International Publishing. (Cited on page 188.)

[180] Panos Vassiliadis. A survey of extract-transform-load technology. *IJDWM*, 5(3):1–27, 2009. (Cited on page 21.)

[181] Henricus M. W. Verbeek, Twan Basten, and Wil M. P. van der Aalst. Diagnosing workflow processes using woflan. *The Computer Journal*, 44(4):246–279, 2001. (Cited on page 105.)

[182] Thomas Vogelgesang. Improving interactivity in multidimensional process mining: The interactive pmcube explorer tool. In *Proceedings of the BPM Demo Track 2017 Co-located with the 15th International Conference on Business Process Management (BPM 2017)*, volume 1920 of *CEUR Workshop Proceedings*, pages 1–5. CEUR-WS.org, 2017. (Cited on page 53.)

[183] Thomas Vogelgesang, Hans-Jürgen Appelrath, et al. Multidimensional process mining: a flexible analysis approach for health services research. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 17–22. ACM, 2013. (Cited on page 53.)

[184] Thomas Vogelgesang and Hans-Jürgen Appelrath. Pmcube: A data-warehouse-based approach for multidimensional process mining. In Manfred Reichert and Hajo A. Reijers, editors, *Business Process Management Workshops: BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers*, pages 167–178, Cham, 2016. Springer International Publishing. (Cited on pages 52 and 53.)

[185] Jianmin Wang, Raymond K. Wong, Jianwei Ding, Qinlong Guo, and Lijie Wen. Efficient Selection of Process Mining Algorithms. *IEEE Transactions on Services Computing*, 6(4):484–496, 2013. (Cited on page 188.)

[186] Ingo H. C. Wassink, Paul E. van der Vet, Katy Wolstencroft, Pieter B. T. Neerincx, Marco Roos, Han Rauwerda, and Timo M. Breit. Analysing scientific workflows: Why workflows not only connect web services. In *2009 IEEE Congress on Services, Part I, SERVICES I 2009, Los Angeles, CA, USA, July 6-10, 2009*, pages 314–321, 2009. (Cited on page 84.)

[187] Philip Weber, Behzad Bordbar, and Peter Tino. A Framework for the Analysis of Process Mining Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(2):303–317, 2013. (Cited on page 188.)

[188] Jochen De Weerdt, Seppe vanden Broucke, Jan Vanthienen, and Bart Baesens. Active Trace Clustering for Improved Process Discovery. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2708–2720, 2013. (Cited on page 159.)

[189] Anton J. M. M. Weijters and Wil M. P. van der Aalst. Rediscovering workflow models from event-based data using Little Thumb. *Integrated*

Computer-Aided Engineering, 10(2):151–162, 2003. (Cited on pages 94, 104, 190, 203, and 205.)

[190] Bernard L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947. (Cited on page 131.)

[191] Alexander Wickert and Anna-Lena Lamprecht. jABCstats: An extensible process library for the empirical analysis of jABC workflows. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, volume 8803 of *Lecture Notes in Computer Science*, pages 449–463. Springer Berlin Heidelberg, 2014. (Cited on page 84.)

[192] David Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. (Cited on page 124.)

[193] Moe T. Wynn, Erik Poppe, Jingxin Xu, Arthur H. M. ter Hofstede, Ross Brown, Azzurra Pini, and Wil M. P. van der Aalst. Processprofiler3d: A visualisation framework for log-based process performance comparison. *Decision Support Systems*, 100:93 – 108, 2017. Smart Business Process Management. (Cited on page 123.)

[194] Reng Zeng, Xudong He, Jiafei Li, Zheng Liu, and Wil M. P. van der Aalst. A method to build and analyze scientific workflows from provenance through process mining. In *3rd Workshop on the Theory and Practice of Provenance, TaPP'11, Heraklion, Crete, Greece, June 20-21, 2011*, 2011. (Cited on page 84.)

[195] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. *An Overview of Concept Drift Applications*, pages 91–114. Springer International Publishing, Cham, 2016. (Cited on page 224.)

Summary

Comparative Process Mining: Analyzing Variability in Process Data

Modern organizations may have a large number of processes with different characteristics. Most of them are supported by information systems ranging from excel sheets to ERP systems. Such systems leave a data footprint that consists of recorded executions of processes i.e., event data.

Process mining is a research discipline that is concerned with discovering, monitoring and improving real processes by extracting knowledge from event data readily available in today's systems. Process mining supports the extraction of insights from data about the overall and inner behavior contained in any given process. Hundreds of different process mining techniques have been proposed in literature.

In real-life, business processes are not static: They must adapt to constant environment changes (e.g., customer preferences, legal regulations, new competitors). Like any live species, organizations (and their business processes) also evolve according to Darwinian evolution: The best to adapt is the one that thrives. It is not uncommon for organizations that the same business process has to adapt to different contexts simultaneously, which leads to variability in the process: different "variants" are born from the adaptation of the process to each context. In many scenarios, splitting a process into variants can effectively reduce its variability (hence, its complexity), making them easier to analyze. It also enables many types of analysis e.g., comparing the different variants of the process in order to identify the best practices and detect differences and similarities between variants.

This thesis addresses the problem of analyzing process variability by proposing techniques and tools that use event data to identify variants within a pro-

cess, split them, compare them, and automate their analysis. We propose a technique to identify process variants from process event data that guarantees statistically significant differences between them. We also propose a technique to split process event data into process variants identified before. In order to compare the process variants found so far, we propose a technique that finds and pinpoints the statistically significant differences between them, so that they can be analyzed. Additionally, we proposed a technique to express and automate process mining experiments in a standardized, transparent, repeatable way. Finally, this thesis adds to the body of scientific knowledge by the application of the aforementioned techniques in real scenarios: working with large companies that shared their data with us and actually used the results reported in this thesis for concrete process improvements.

In summary, this thesis proposes to address variability in process data starting by establishing the notion of process variants as a core abstraction to represent the adaptation of the process to different situations and contexts. Then, this thesis proposes several techniques to identify, split and compare them and automate their analysis. Finally, several case-study applications prove that these techniques work using real-life data from known companies.

Acknowledgments

During the (long) writing process of this thesis, several people had an impact on me and my work.

First of all, I want to thank my promotor Wil van der Aalst. Under his tough but excellent guidance, I acquired several skills that are invaluable nowadays in my life. He also mentored me personally and, in a way, shaped me into the researcher that I am today.

I want to thank my co-promotor Hajo Reijers for his unique personal touch that helped me push through some difficult situations, and also a special thanks to Massimiliano de Leoni, who was my daily supervisor over the course of two years. We had some heated discussions, but ultimately his supervision resulted in one of the most productive periods of my life.

I also wanted to thank Eric for his infinite patience and support towards me related to ProM coding. Without the support of the secretariat (thanks to Riet, Jose and specially Ine: you were always there when I needed something) this whole process would have been much more difficult, so many thanks to you.

Any PhD student knows that work is only half of a student's life. I would like to mention a very special group of people: my fellow PhD students with which I shared so many discussions, tears, laughs, drinks, stories, bbqs and wild karaoke nights: Bas, Edu, Maikel, Niek, Alok, Felix, Sander, Xixi, Shiva, Alifah, Bart, Mohammadreza, Elham, Dennis, Cong, Joos, Marcus, Long, Marie, Nour, Petar, Rafal, Shengnan, Vadim and many others.

Furthermore, I would like to thank my fellow musicians from the city of Eindhoven: Owen, Raquel, Pauraig, Llaima, Akash, Ashok, Antoine, Waldo, Mohsen, Anita, Jack, Javier, Mikele, Carmine, Davide, Jamie, and all the great people I met at Music with Strangers. Music saved me so many times, mostly through musicians like you.

Finally, I would like to thank my family for their support and love during

good and bad times. My heart goes with you always.

Alfredo José Bolt Iriondo
Eindhoven, January 2023

Curriculum Vitae

Alfredo José Bolt Iriondo was born on 28-12-1985 in Santiago, Chile. After finishing Industrial Engineering in 2010 at Pontificia Universidad Católica de Chile in Santiago, Chile, he studied a Master's in Computer Science at Pontificia Universidad Católica de Chile in Santiago, Chile. In 2012 he graduated with maximum distinction within the Department of Computer Science on the topic of Process Mining. From 2012 to 2014 he worked as a Consultant in Business Process Management. From 2014 he started a PhD project at Eindhoven University of Technology at Eindhoven, The Netherlands, of which the results are presented in this dissertation. Since 2018 he is employed as an Assistant Professor at Universidad Finis Terrae in Santiago, Chile.

List of Publications

Alfredo José Bolt Iriondo has the following publications:

Journals

- Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. Scientific workflows for process mining: building blocks, scenarios, and implementation. *International Journal on Software Tools for Technology Transfer*, 18(6):607–628, Nov 2016
- Alfredo Bolt, Massimiliano de Leoni, and Wil M.P. van der Aalst. Process variant comparison: Using event logs to detect differences in behavior and business rules. *Information Systems*, 74:53 – 66, 2018. Information Systems Engineering: selected papers from CAiSE 2016
- Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8:269–284, 2019

Proceedings and Congres Contributions

- Alfredo Bolt and Wil M. P. van der Aalst. Multidimensional process mining using process cubes. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 102–116, Cham, 2015. Springer International Publishing
- Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. A visual approach to spot statistically-significant differences in event logs based on process metrics. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering*, pages 151–166, Cham, 2016. Springer International Publishing
- Alfredo Bolt, Wil M. P. van der Aalst, and Massimiliano de Leoni. Finding process variants in event logs. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 45–52, Cham, 2017. Springer International Publishing

- Alifah Syamsiyah, Alfredo Bolt, Long Cheng, Bart F. A. Hompes, R. P. Jagadeesh Chandra Bose, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Business process comparison: A methodology and case study. In Witold Abramowicz, editor, *Business Information Systems*, pages 253–267, Cham, 2017. Springer International Publishing

- Mohammadreza Fani Sani, Wil van der Aalst, Alfredo Bolt, and Javier García-Algarra. Subgroup discovery in process mining. In Witold Abramowicz, editor, *Business Information Systems*, pages 237–252, Cham, 2017. Springer International Publishing

- Sebastiaan J. van Zelst, Alfredo Bolt , and Boudewijn F. van Dongen. Tuning alignment computation: an experimental evaluation. CEUR Workshop Proceedings, pages 6–20, 2017. Workshop on Algorithms and theories for the analysis of event data (ATAED2017), 26-27 June 2017, Zaragoza, Spain

- Alfredo Bolt, Massimiliano De Leoni, Wil M.P. van der Aalst, and Pierre Gorissen. Exploiting process cubes, analytic workflows and process mining for business process reporting: A case study in education. In P. Ceravolo and S. Rinderle-Ma, editors, *Data-Driven Process Discovery and Analysis (SIMPDA 2015), December 9-11, 2015, Vienna, Austria*, CEUR Workshop Proceedings, pages 33–47. CEUR-WS.org, 2015. 5th International Symposium on Data-Driven Process Discovery and Analysis, SIMPDA 2015 ; Conference date: 09-12-2015 Through 11-12-2015

- A.J. Bolt Iriondo, M. de Leoni, W.M.P. van der Aalst, and P. Gorissen. Business process reporting using process mining, analytic workflows and process cubes: A case study in education. In C. Paolo and R.-M. Stefanie, editors, *Data-Driven Process Discovery and Analysis*, Lecture Notes in Business Information Processing, pages 28–53, Germany, 2017. Springer. 5th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2015, SIMPDA 2015 ; Conference date: 09-12-2015 Through 11-12-2015

- Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. *Computing alignments of event data and process models*, pages 1–26. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Germany, January 2018

- Iulia. Efremova, Alejandro Montes Garcia, Alfredo. Bolt, and Toon G.K. Calders. Who are my ancestors? : retrieving family relationships from historical texts. In P. Braslavski , I. Markov, P. Pardalos, Y. Volkovich, D.I. Ignatov , S. Koltsov, and O. Koltsova, editors, *Information Retrieval*, Communications in Computer and Information Science, pages 121–129, Germany, 2015. Springer

Master Thesis

- Alfredo Bolt and Marcos Sepúlveda. Process remaining time prediction using query catalogs. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, pages 54–65, Cham, 2014. Springer International Publishing

Technical Reports (Non-Refereed)

- Wil MP van der Aalst, Alfredo Bolt, and Sebastiaan J van Zelst. Rapidprom: mine your processes and not just your data. *arXiv preprint arXiv:1703.03740*, 2017

- Toon Jouck, Alfredo Bolt, Benoît Depaire, Massimiliano de Leoni, and Wil MP van der Aalst. An integrated framework for process discovery algorithm evaluation. Technical report, https://arxiv.org/abs/1806.07222, 2018

SIKS Dissertations

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 (TU/e), *Compositional Design and Verification of Component-Based Information Systems.*

2011-04 Hado Philip van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms.*

2011-05 Bas van de Raadt (VUA), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.*

2011-06 Yiwen Wang (TU/e), *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 (TiU), *Cloud Content Contention.*

2011-11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*

2011-12 Carmen Bratosin (TU/e), *Grid Architecture for Distributed Process Mining.*

2011-13 Xiaoyu Mao (TiU), *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 (VUA), *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 (VUA), *Analysis and Validation of Models for Trust Dynamics.*

2011-26 Matthijs Aart Pontier (VUA), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*

2011-27 Aniel Bhulai (VUA), *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 (TU/e), *Discrimination-aware Classification.*

2011-30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*

2012

2012-01 Terry Kakeeto (TiU), *Relationship Marketing for SMEs in Uganda.*

2012-02 Muhammad Umair (VUA), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*

2012-03 Adam Vanya (VUA), *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 (OUN), *Awareness Support for Knowledge Workers in Research Networks.*

2012-07 Rianne van Lambalgen (VUA), *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 (TU/e), *Towards a Generic Distributed Adaptive Hypermedia Environment.*

2012-11 J.C.B. Rantham Prabhakara (TU/e), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*

2012-12 Kees van der Sluijs (TU/e), *Model Driven Design and Data Integration in Semantic Web Information Systems.*

2012-13 Suleman Shahid (TiU), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*

2012-14 Evgeny Knutov(TU/e), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*

2012-15 Natalie van der Wal (VUA), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*

2012-16 Fiemke Both (VUA), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*

2012-17 Amal Elgammal (TiU), *Towards a Comprehensive Framework for Business Process Compliance.*

2012-18 Eltjo Poort (VUA), *Improving Solution Architecting Practices.*

2012-19 Helen Schonenberg (TU/e), *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 (TiU), *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 Gürkök (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*

2012-28 Nancy Pascall (TiU), *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 (VUA), *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 (VUA), *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 (TiU), *Information Access for SMEs in Indonesia.*

2012-41 Sebastian Kelle (OUN), *Game Design Patterns for Learning.*

2012-42 Dominique Verpoorten (OUN), *Reflection Amplifiers in self-regulated Learning.*

2012-43 Withdrawn, *List of dissertations 2012.*

2012-44 Anna Tordai (VUA), *On Combining Alignment Techniques.*

2012-45 Benedikt Kratz (TiU), *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 (TU/e), *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), *Ontology driven Enterprise Information Systems Engineering.*

2012-51 Jeroen de Jong (TUD), *Heuristics in Dynamic Scheduling; 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 (VUA), *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 (TiU), *Quantifying Individual Player Differences.*

2013-08 Robbert-Jan Merk (VUA), *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 (TiU), *A Unified Modeling Framework for Service Design..*

2013-11 Evangelos Pournaras (TUD), *Multi-level Reconfigurable Self-organization in Overlay Services.*

2013-12 Maryam Razavian (VUA), *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 (VUA), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*

2013-18 Jeroen Janssens (TiU), *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 (TiU), *Text-to-text generation by monolingual machine translation.*

2013-22 Tom Claassen (RUN), *Causal Discovery and Logic.*

2013-23 Patricio de Alencar Silva (TiU), *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 (TU/e), *Resource-Aware Business Process Management: Analysis and Support.*

2013-31 Dinh Khoa Nguyen (TiU), *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 Pro-*

fessional 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.*

2013-36 Than Lam Hoang (TU/e), *Pattern Mining in Data Streams.*

2013-37 Dirk Börner (OUN), *Ambient Learning Displays.*

2013-38 Eelco den Heijer (VUA), *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 Léon 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 (TU/e), *Aligning Observed and Modeled Behavior.*

2014-08 Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints.*

2014-09 Philip Jackson (TiU), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.*

2014-10 Ivan Salvador Razo Zapata (VUA), *Service Value Networks.*

2014-11 Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support.*

2014-12 Willem van Willigen (VUA), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control.*

2014-12 Arlette van Wissen (VUA), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.*

2014-14 Yangyang Shi (TUD), *Language Models With Meta-information.*

2014-15 Natalya Mogles (VUA), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*

2014-16 Krystyna Milian (VUA), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*

2014-17 Kathrin Dentler (VUA), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*

2014-18 Mattijs Ghijsen (UvA), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*

2014-19 Vinicius Ramos (TU/e), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*

2014-20 Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*

2014-21 Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments.*

2014-22 Marieke Peeters (UU), *Personalized Educational Games - Developing agent-supported scenario-based training.*

2014-23 Eleftherios Sidirourgos (UvA/CWI), *Space Efficient Indexes for the Big Data Era.*

2014-24 Davide Ceolin (VUA), *Trusting Semi-structured Web Data.*

2014-25 Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*

2014-26 Tim Baarslag (TUD), *What to Bid and When to Stop.*

2014-27 Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*

2014-28 Anna Chmielowiec (VUA), *Decentralized k-Clique Matching.*

2014-29 Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*

2014-30 Peter de Cock (TiU), *Anticipating Criminal Behaviour.*

2014-31 Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support.*

2014-32 Naser Ayat (UvA), *On Entity Resolution in Probabilistic Data.*

2014-33 Tesfa Tegegne (RUN), *Service Discovery in eHealth.*

2014-34 Christina Manteli (VUA), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems..*

2014-35 Joost van Oijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach.*

2014-36 Joos Buijs (TU/e), *Flexible Evolutionary Algorithms for Mining Structured Process Models..*

2014-37 Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying.*

2014-38 Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing..*

2014-39 Jasmina Maric (TiU), *Web Communities, Immigration, and Social Capital.*

2014-40 Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education..*

2014-41 Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text.*

2014-42 Carsten Eijckhof (CWI/TUD), *Contextual Multidimensional Relevance Models.*

2014-43 Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments.*

2014-44 Paulien Meesters (TiU), *Intelligent Blauw: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden..*

2014-45 Birgit Schmitz (OUN), *Mobile Games for Learning: A Pattern-Based Approach.*

2014-46 Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity.*

2014-47 Shangsong Liang (UvA), *Fusion and Diversification in Information Retrieval.*

2015

2015-01 Niels Netten (UVA), *Machine Learning for Relevance of Information in Crisis Response.*

2015-02 Faiza Bukhsh (UVT), *Smart auditing: Innovative Compliance Checking in Customs Controls.*

2015-03 Twan van Laarhoven (RUN), *Machine learning for network data.*

2015-04 Howard Spoelstra (OUN), *Collaborations in Open Learning Environments.*

2015-05 Christoph Bösch (UT), *Cryptographically Enforced Search Pattern Hiding.*

2015-06 Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes.*

2015-07 Maria-Hendrike Peetz (UVA), *Time-Aware Online Reputation Analysis.*

2015-08 Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions.*

2015-09 Randy Klaassen (UT), *HCI Perspectives on Behavior Change Support Systems.*

2015-10 Henry Hermans (OUN), *OpenU: design of an integrated system to support lifelong learning.*

2015-11 Yongming Luo (TUE), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins.*

2015-12 Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks.*

2015-13 Giuseppe Procaccianti (VU), *Energy-Efficient Software.*

2015-14 Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations.*

2015-15 Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation.*

2015-16 Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork.*

2015-17 André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs.*

2015-18 Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories.*

2015-19 Bernardo Tabuenca (OUN), *Ubiquitous Technology for Lifelong Learners.*

2015-20 Loïs Vanhée (UU), *Using Culture and Values to Support Flexible Coordination Using Culture and Values to Support Flexible Coordination.*

2015-21 Sibren Fetter (OUN), *Using Culture and Values to Support Flexible CoordinationUsing Peer-Support to Expand and Stabilize Online Learning.*

2015-22 Zhemin Zhu (UT), *Co-occurrence Rate Networks - Towards separate training for undirected graphical models.*

2015-23 Luit Gazendam (VU), *Using Culture and Values to Support Flexible CoordinationCataloguer Support in Cultural Heritage.*

2015-24 Richard Berendsen (UVA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation.*

2015-25 Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection.*

2015-26 Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure.*

2015-27 Sándor Héman (CWI), *Updating compressed colomn stores.*

2015-28 Janet Bagorogoza (TiU), *Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO.*

2015-29 Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains.*

2015-30 Kiavash Bahreini (OU), *Real-time Multimodal Emotion Recognition in E-Learning.*

2015-31 Yakup Koç (TUD), *On the robustness of Power Grids.*

2015-32 Jerome Gard (UL), *Corporate Venture Management in SMEs.*

2015-33 Frederik Schadd (UM), *Ontology Mapping with Auxiliary Resources.*

2015-34 Victor de Graaff (UT), *Geosocial Recommender Systems.*

2015-35 Jungxao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.*

2016

2016-01 Syed Saiden Abbas (RUN), *Recognition of Shapes by Humans and Machines.*

2016-02 Michiel Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow.*

2016-03 Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support.*

2016-04 Laurens Rietveld (VU), *Publishing and Consuming Linked Data.*

2016-05 Evgeny Sherkhonov (UVA), *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers.*

2016-06 Michel Wilson (TUD), *Robust scheduling in an uncertain environment.*

2016-07 Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training.*

2016-08 Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data.*

2016-09 Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts.*

2016-10 George Karafotias (VU), *Parameter Control for Evolutionary Algorithms.*

2016-11 Anne Schuth (UVA), *Search Engines that Learn from Their Users.*

2016-12 Max Knobbout (UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems.*

2016-13 Nana Baah Gyan (VU), *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach.*

2016-14 Ravi Khadka (UU), *Revisiting Legacy Software System Modernization.*

2016-15 Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments.*

2016-16 Guangliang Li (UVA), *Socially Intelligent Autonomous Agents that Learn from Human Reward.*

2016-17 Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms.*

2016-18 Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web.*

2016-19 Julia Efremova (TUE), *Mining Social Structures from Genealogical Data.*

2016-20 Daan Odijk (VU), *Context & Semantics in News & Web Search.*

2016-21 Alejandro Moreno Celleri (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground.*

2016-22 Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems.*

2016-23 Fei Cai (UVA), *Query Auto Completion in Information Retrieval.*

2016-24 Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach.*

2016-25 Y. Kiseleva (TUE), *Using Contextual Information to Understand Searching and Browsing Behavior.*

2016-26 Dilhan J. Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management*

Domains.

2016-27 Wen Li (TUD), *Understanding Geo-spatial Information on Social Media.*

2016-28 Mingxin Zhang (TUD), *Large-scale agent-based social simulation: A study on epidemic prediction and control.*

2016-29 Nicolas Höning (TUD), *Understanding Geo-spatial Information on Social Media.*

2016-30 Ruud Mattheij (UVT), *The Eyes Have IT.*

2016-31 Mohammad Khelghati (UT), *Deep web content monitoring.*

2016-32 Eelco Vriezekolk (UVT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations.*

2016-33 Peter Bloem (UVA), *Single Sample Statistics, exercises in learning from just one example.*

2016-34 Dennis Schunselaar (TUE), *Configurable Process Trees: Elicitation, Analysis, and Enactment.*

2016-35 Zhaochun Ren (UVA), *Monitoring Social Media: Summarization, Classification and Recommendation.*

2016-36 Daphne Karreman (UT), *Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies.*

2016-37 Giovanni Sileno (UVA), *Aligning Law and Action - a conceptual and computational inquiry.*

2016-38 Andrea Minuto (UT), *Materials that matter - Smart Materials meet Art & Interaction Design.*

2016-39 Merijn Bruijnes, *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect.*

2016-40 Christian Detweiler (TUD), *Accounting for Values in Design.*

2016-41 Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance.*

2016-42 Spyros Martzoukos (UVA), *Combinatorial and compositional aspects of bilingual aligned corpora.*

2016-43 Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From Theory to Practice.*

2016-44 Thibault Sellam (UVA), *Automatic assistants for database exploration.*

2016-45 Bram van Laar (UT), *Experiencing Brain-Computer Interface Control.*

2016-46 Jorge Gallego Perez (UT), *Robots to Make you Happy.*

2016-47 Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks.*

2016-48 Tanja Buttler (TUD), *Collecting Lessons Learned.*

2016-49 Gleb Polevoy (TUD), *Participation and Interaction in Projects: A Game-Theoretic Analysis.*

2016-50 Yan Wang (UVT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains.*

2017

2017-01 Jan-Jaap Oerlemans (UL), *Investigating Cybercrime.*

2017-02 Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation.*

2017-03 Daniël Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines.*

2017-04 Mrunal Gawade (CWI), *Multi-core parallelism in a column-store.*

2017-05 Mahdieh Shadi (UVA), *Collaboration Behavior; Enhancement in Co-development.*

2017-06 Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search.*

2017-07 Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly.*

2017-08 Rob Konijn (VU), *Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery.*

2017-09 Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text.*

2017-10 Robby van Delden (UT), *(Steering) Interactive Play Behavior.*

2017-11 Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment.*

2017-12 Sander Leemans (TUE), *Robust Process Mining with Guarantees.*

2017-13 Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technology.*

2017-14 Shoshannah Tekofsky (UVT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior.*

2017-15 Peter Berck (RUN), *Memory-Based Text Correction.*

2017-16 Aleksandr Chuklin (UVA), *Understanding and Modeling Users of Modern Search Engines.*

2017-17 Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution.*

2017-18 Ridho Reinanda (UVA), *Entity Associations for Search.*

2017-19 Jeroen Vuurens (TUD), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval.*

2017-20 Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility.*

2017-21 Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds).*

2017-22 Sara Magliacane (VU), *Logics for causal inference under uncertainty.*

2017-23 David Graus (UVA), *Entities of Interest - Discovery in Digital Traces.*

2017-24 Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning.*

2017-25 Veruska Zamborlini (VUA), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search.*

2017-26 Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch.*

2017-27 Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors.*

2017-28 John Klein (VU), *Architecture Practices for Complex Contexts.*

2017-29 Adel Alhuraibi (UVT), *From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT.*

2017-30 Wilma Latuny (UVT), *The Power of Facial Expressions.*

2017-31 Ben Ruijl (UL), *Advances in computational methods for QFT calculations.*

2017-32 Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives.*

2017-33 Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity.*

2017-34 Maren Scheffel (OUN), *The Evaluation Framework for Learning Analytics.*

2017-35 Martine de Vos (VU), *Interpreting natural science spreadsheets.*

2017-36 Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging.*

2017-37 Alejandro Montes Garcia (TUE), *WiBAF: A Within Browser Adaptation Framework that Enables Control*

over Privacy.

2017-38 Abdullah Kayal (TUD), *Normative Social Applications.*

2017-39 Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR.*

2017-40 Altaf Hussain Abro (VUA), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems.*

2017-41 Adnan Manzoor (VUA), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle.*

2017-42 Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets.*

2017-43 Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval.*

2017-44 Garm Lucassen (UU), *Understanding User Stories - Computational Linguistics in Agile Requirements Engineering.*

2017-45 Bas Testerink (UU), *Decentralized Runtime Norm Enforcement.*

2017-46 Jan Schneider (OU), *Sensor-based Learning Support.*

2017-47 Yie Yang (TUD), *Crowd Knowledge Creation Acceleration.*

2017-48 Angel Suarez (OU), *Collaborative inquiry-based learning.*

2018

2018-01 Han van der Aa (VU), *Comparing and Aligning Process Representations.*

2018-02 Felix Mannhardt (TUE), *Multi-perspective Process Mining.*

2018-03 Steven Bosems (UT), *Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction.*

2018-04 Jordan Janeiro (TUD), *Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks.*

2018-05 Hugo Huurdeman (UVA), *Supporting the Complex Dynamics of the Information Seeking Process.*

2018-06 Dan Ionita (UT), *Model-Driven Information Security Risk Assessment of Socio-Technical Systems.*

2018-07 Jieting Luo (UU), *A formal account of opportunism in multi-agent systems.*

2018-08 Rick Smetsers (RUN), *Advances in Model Learning for Software Systems.*

2018-09 Xu Xie (TUD), *Data Assimilation in Discrete Event Simulations.*

2018-10 Julienka Mollee (VUA), *Moving forward: supporting physical activity behavior change through intelligent technology.*

2018-11 Mahdi Sargolzaei (UVA), *Enabling Framework for Service-oriented Collaborative Networks.*

2018-12 Xixi Lu (TUE), *Using behavioral context in process mining.*

2018-13 Seyed Amin Tabatabaei (VUA), *Computing a Sustainable Future: Exploring the added value of computational models for increasing the use of renewable energy in the residential sector.*

2018-14 Bart Joosten (UVT), *Detecting Social Signals with Spatiotemporal Gabor Filters.*

2018-15 Naser Davarzani (UM), *Biomarker discovery in heart failure.*

2018-16 Jaebok Kim (UT), *Automatic recognition of engagement and emotion in a group of children.*

2018-17 Jianpeng Zhang (TUE), *On Graph Sample Clustering.*

2018-18 Henriette Nakad (UL), *De Notaris en Private Rechtspraak.*

2018-19 Minh Duc Pham (VUA), *Emergent relational schemas for RDF.*

2018-20 Manxia Liu (RUN), *Time and Bayesian Networks.*

2018-21 Aad Slootmaker (OU), *EMERGO: a generic platform for authoring and playing scenario-based serious games.*

2018-22 Eric Fernandes de Mello Araújo (VUA), *Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks.*

2018-23 Kim Schouten (EUR), *Semantics-driven Aspect-Based Sentiment Analysis.*

2018-24 Jered Vroon (UT), *Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots.*

2018-25 Riste Gligorov (VUA), *Serious Games in Audio-Visual Collections.*

2018-26 Roelof de Vries (UT), *Theory-Based And Tailor-Made: Motivational Messages for Behavior Change Technology.*

2018-27 Maikel Leemans (TUE), *Hierarchical Process Mining for Scalable Software Analysis.*

2018-28 Christian Willemse (UT), *Social Touch Technologies: How they feel and how they make you feel.*

2018-29 Yu Gu (UVT), *Emotion Recognition from Mandarin Speech.*

2018-30 Wouter Beek (VU), *The K in semantic web stands for knowledge: scaling semantics to the web.*

2019

2019-01 Rob van Eijk (UL), *Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification.*

2019-02 Emmanuelle Beauxis- Aussalet (CWI, UU), *Statistics and Visualizations for Assessing Class Size Uncertainty.*

2019-03 Eduardo Gonzalez Lopez de Murillas (TUE), *Process Mining on Databases: Extracting Event Data from Real Life Data Sources.*

2019-04 Ridho Rahmadi (RUN), *Finding stable causal structures from clinical data.*

2019-05 Sebastiaan van Zelst (TUE), *Process Mining with Streaming Data.*

2019-06 Chris Dijkshoorn (VU), *Nichesourcing for Improving Access to Linked Cultural Heritage Datasets.*

2019-07 Soude Fazeli (TUD), *Recommender Systems in Social Learning Platforms.*

2019-08 Frits de Nijs (TUD), *Resource-constrained Multi-agent Markov Decision Processes.*

2019-09 Fahimeh Alizadeh Moghaddam (UVA), *Self-adaptation for energy efficiency in software systems.*

2019-10 Qing Chuan Ye (EUR), *Multi-objective Optimization Methods for Allocation and Prediction.*

2019-11 Yue Zhao (TUD), *Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs.*

2019-12 Jacqueline Heinerman (VU), *Better Together.*

2019-13 Guanliang Chen (TUD), *MOOC Analytics: Learner Modeling and Content Generation.*

2019-14 Daniel Davis (TUD), *Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses.*

2019-15 Erwin Walraven (TUD), *Planning under Uncertainty in Constrained and Partially Observable Environments.*

2019-16 Guangming Li (TUE), *Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models.*

2019-17 Ali Hurriyetoglu (RUN), *Extracting actionable information from microtexts.*

2019-18 Gerard Wagenaar (UU), *Artefacts in Agile Team Communication.*

2019-19 Vincent Koeman (TUD), *Tools for Developing Cognitive Agents.*

2019-20 Chide Groenouwe (UU), *Fostering technically augmented human collective intelligence.*

2019-21 Cong Liu (TUE), *Software Data Analytics: Architectural Model Discovery and Design Pattern Detection.*

2019-22 Martin van den Berg (VU), *Improving IT Decisions with Enterprise Architecture.*

2019-23 Qin Lin (TUD), *Intelligent Control Systems: Learning, Interpreting, Verification.*

2019-24 Anca Dumitrache (VU), *Truth in Disagreement- Crowdsourcing Labeled Data for Natural Language Processing.*

2019-25 Emiel van Miltenburg (VU), *Pragmatic factors in (automatic) image description.*

2019-26 Prince Singh (UT), *An Integration Platform for Synchromodal Transport.*

2019-27 Alessandra Antonaci (OUN), *The Gamification Design Process applied to (Massive) Open Online Courses.*

2019-28 Esther Kuindersma (UL), *Cleared for take-off:Game-based learning to prepare airline pilots for critical situations.*

2019-29 Daniel Formolo (VU), *Using virtual agents for simulation and training of social skills in safety-critical circumstances.*

2019-30 Vahid Yazdanpanah (UT), *Multiagent Industrial Symbiosis Systems.*

2019-31 Milan Jelisavcic (VU), *Alive and Kicking: Baby Steps in Robotics.*

2019-32 Chiara Sironi (UM), *Monte-Carlo Tree Search for Artificial General Intelligence in Games.*

2019-33 Anil Yaman (TUE), *Evolution of Biologically Inspired Learning in Artificial Neural Networks.*

2019-34 Negar Ahmadi (TUE), *EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES.*

2019-35 Lisa Facey-Shaw (OUN), *Gamification with digital badges in learning programming.*

2019-36 Kevin Ackermans (OUN), *Designing Video-Enhanced Rubrics to Master Complex Skills.*

2019-37 Jian Fang (TUD), *Database Acceleration on FPGAs.*

2019-38 Akos Kadar (OUN), *Learning visually grounded and multilingual representations.*

2020

2020-01 Armon Toubman (UL), *Calculated Moves: Generating Air Combat Behaviour.*

2020-02 Marcos de Paula Bueno (UL), *Unraveling Temporal Processes using Probabilistic Graphical Models.*

2020-03 Mostafa Deghani (UvA), *Learning with Imperfect Supervision for Language Understanding.*

2020-04 Maarten van Gompel (RUN), *Context as Linguistic Bridges.*

2020-05 Yulong Pei (TUE), *On local and global structure mining.*

2020-06 Preethu Rose Anish (UT), *Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support.*

2020-07 Wim van der Vegt (OUN), *Towards a software architecture for reusable game components.*

2020-08 Ali Mirsoleimani (UL), *Structured Parallel Programming for Monte Carlo Tree Search.*

2020-09 Myriam Traub (UU), *Measuring Tool Bias & Improving Data Quality for Digital Humanities Research.*

2020-10 Alifah Syamsiyah (TUE), *In-database Preprocessing for Process Mining.*

2020-11 Sepideh Mesbah (TUD), *Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models.*

2020-12 Ward van Breda (VU), *Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment.*

2020-13 Marco Virgolin (CWI/ TUD), *Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming.*

2020-14 Mark Raasveldt (CWI/UL), *Integrating Analytics with Relational Databases.*

2020-15 Georgiadis Konstantinos (OU), *Smart CAT: Machine Learning for Configurable Assessments in Serious Games.*

2020-16 Ilona Wilmont (RUN), *Cognitive Aspects of Conceptual Modelling.*

2020-17 Daniele Di Mitri (OU), *The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences.*

2020-18 Georgios Methenitis (TUD), *Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems.*

2020-19 Guido van Capelleveen (UT), *Industrial Symbiosis Recommender Systems.*

2020-20 Albert Hankel (VU), *Embedding Green ICT Maturity in Organisations.*

2020-21 Karine da Silva Miras de Araujo (VU), *Where is the robot?: Life as it could be.*

2020-22 Maryam Masoud Khamis (RUN), *Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar.*

2020-23 Rianne Conijn (UT), *The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging.*

2020-24 Lenin da Nóbrega Medeiros (VUA/RUN), *How are you feeling, human? Towards emotionally supportive chatbots.*

2020-25 Xin Du (TUE), *The Uncertainty in Exceptional Model Mining.*

2020-26 Krzysztof Leszek Sadowski (UU), *GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization.*

2020-27 Ekaterina Muravyeva (TUD), *Personal data and informed consent in an educational context.*

2020-28 Bibeg Limbu (TUD), *Multimodal interaction for deliberate practice: Training complex skills with augmented reality.*

2020-29 Ioan Gabriel Bucur (RUN), *Being Bayesian about Causal Inference.*

2020-30 Bob Zadok Blok (UL), *Creatief, Creatieve, Creatiefst.*

2020-31 Gongjin Lan (VU), *Learning Better: From Baby to Better.*

2020-32 Jason Rhuggenaath (TUE), *Revenue management in online markets: pricing and online advertising.*

2020-33 Rick Gilsing (TUE), *Supporting service-dominant business model evaluation in the context of business model innovation.*

2020-34 Anna Bon (MU), *Intervention or Collaboration? Redesigning Information and Communication Technologies for Development.*

2020-35 Siamak Farshidi (UU), *Multi-Criteria Decision-Making in Software Production.*

2021

2021-01 Francisco Xavier Dos Santos Fonseca (TUD), *Location-based Games for Social Interaction in Public Space.*

2021-02 Rijk Mercuur (TUD), *Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models.*

2021-03 Seyyed Hadi Hashemi (UVA), *Modeling Users Interacting with Smart Devices.*

2021-04 Ioana Jivet (OU)), *The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning.*

2021-05 Davide Dell'Anna (UU), *Data-Driven Supervision of Autonomous Systems.*

2021-06 Daniel Davison (UT), *"Hey robot, what do you think?" How children learn with a social robot.*

2021-07 Armel Lefebvre (UU), *Research data management for open science.*

2021-08 Nardie Fanchamps (OU), *The Influence of Sense-Reason-Act Programming on Computational Thinking.*

2021-09 Cristina Zaga (UT), *The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play.*

2021-10 Quinten Meertens (UvA), *Misclassification Bias in Statistical Learning.*

2021-11 Anne van Rossum (UL), *Nonparametric Bayesian Methods in Robotic Vision.*

2021-12 Lei Pi (UL), *External Knowledge Absorption in Chinese SMEs.*

2021-13 Bob R. Schadenberg (UT), *Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning.*

2021-14 Negin Samaeemofrad (UL), *Business Incubators: The Impact of Their Support.*

2021-15 Onat Ege Adali (TU/e), *Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm.*

2021-16 Esam A. H. Ghaleb (UM), *Bimodal emotion recognition from audio-visual cues.*

2021-17 Dario Dotti (UM), *Human Behavior Understanding from motion and bodily cues using deep neural networks.*

2021-18 Remi Wieten (UU), *Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks.*

2021-19 Roberto Verdecchia (VU), *Architectural Technical Debt: Identification and Management.*

2021-20 Masoud Mansoury (TU/e), *Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems.*

2021-21 Pedro Thiago Timbó Holanda (CWI), *Progressive Indexes.*

2021-22 Sihang Qiu (TUD), *Conversational Crowdsourcing.*

2021-23 Hugo Manuel Proença (LIACS), *Robust rules for prediction and description.*

2021-24 Kaijie Zhu (TUE), *On Efficient Temporal Subgraph Query Processing.*

2021-25 Eoin Martino Grua (VUA), *The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications.*

2021-26 Benno Kruit (CWI & VU), *Reading the Grid: Extending Knowledge Bases from Human-readable Tables.*

2021-27 Jelte van Waterschoot (UT), *Personalized and Personal Conversations: Designing Agents Who Want to Connect With You.*

2021-28 Christoph Selig (UL), *Understanding the Heterogeneity of Corporate Entrepreneurship Programs.*

2022

2022-01 Judith van Stegeren (UT), *Flavor text generation for role-playing video games.*

2022-02 Paulo da Costa (TU/e), *Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey.*

2022-03 Ali el Hassouni (VUA), *A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare.*

2022-04 Ünal Aksu (UU), *A Cross-Organizational Process Mining Framework.*

2022-05 Shiwei Liu (TU/e), *Sparse Neural Network Training with In-Time Over-Parameterization.*

2022-06 Reza Refaei Afshar (TU/e), *Machine Learning for Ad Publishers in Real Time Bidding.*

2022-07 Sambit Praharaj (OU), *Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics.*

2022-08 Maikel L. van Eck (TU/e), *Process Mining for Smart Product Design.*

2022-09 Oana Andreea Inel (VUA), *Understanding Events: A Diversity-driven Human-Machine Approach.*

2022-10 Felipe Moraes Gomes (TUD), *Examining the Effectiveness of Collaborative Search Engines.*

2022-11 Mirjam de Haas (TiU), *Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring.*

2022-12 Guanyi Chen (UU), *Computational Generation of Chinese Noun Phrases.*

2022-13 Xander Wilcke (VUA), *Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge.*

2022-14 Michiel Overeem (UU), *Evolution of Low-Code Platforms.*

2022-15 Jelmer Jan Koorn (UU), *Work in Process: Unearthing Meaning using Process Mining.*

2022-16 Pieter Gijsbers (TU/e), *Systems for AutoML Research.*

2022-17 Laura van der Lubbe (VUA), *Empowering vulnerable people with serious games and gamification.*

2022-18 Paris Mavromoustakos Blom (TiU), *Player Affect Modelling and Video Game Personalisation.*

2022-19 Bilge Yigit Ozkan (UU), *Cybersecurity Maturity Assessment and Standardisation.*

2022-20 Fakhra Jabeen (VUA), *Dark Side of the Digital World - Computational Analysis of Negative Human Behaviors on Social Media.*

2022-21 Seethu Mariyam Christopher (UM), *Intelligent Toys for Physical and Cognitive Assessments.*

2022-22 Alexandra Sierra Rativa (TiU), *Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations.*

2022-23 Ilir Kola (TUD), *Enabling Social Situation Awareness in Support Agents.*

2022-24 Samaneh Heidari (UU), *Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values.*

2022-25 Anna L.D. Latour (LU), *Optimal decision-making under constraints and uncertainty.*

2022-26 Anne Dirkson (LU), *Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences.*

2022-27 Christos Athanasiadis (UM), *Emotion-aware cross-modal domain adaptation in video sequences.*

2022-28 Onuralp Ulusoy (UU), *Privacy in Collaborative Systems.*

2022-29 Jan Kolkmeier (UT-EEMCS), *From Head Transform to Mind Transplant: Social Interactions in Mixed Reality.*

2022-30 Dean De Leo (CWI), *Analysis of Dynamic Graphs on Sparse Arrays.*

2022-31 Konstantinos Traganos (TU/e), *Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management.*

2022-32 Cezara Pastrav (UU), *Social simulation for socio-ecological systems.*

2022-33 Brinn Hekkelman (CWI/TUD), *Fair Mechanisms for Smart Grid Congestion Management.*

2022-34 Nimat Ullah (VUA), *Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change.*

2023

2023-01 Alfredo Bolt Iriondo (TUE), *Comparative Process Mining: Analyzing Variability in Process Data.*