

RapidProM: Mine Your Processes and Not Just Your Data

Wil M.P. van der Aalst, Alfredo Bolt, Sebastiaan J. van Zelst

March 13, 2017

Abstract

The number of events recorded for operational processes is growing every year. This applies to all domains: from health care and e-government to production and maintenance. Event data are a valuable source of information for organizations that need to meet requirements related to compliance, efficiency, and customer service. Process mining helps to turn these data into real value: by discovering the real processes, by automatically identifying bottlenecks, by analyzing deviations and sources of non-compliance, by revealing the actual behavior of people, etc. Process mining is very different from conventional data mining and machine learning techniques. PROM is a powerful open-source process mining tool supporting hundreds of analysis techniques. However, PROM does not support analysis based on scientific workflows. RAPIDPROM, an extension of RAPIDMINER based on PROM, combines the best of both worlds. Complex process mining workflows can be modeled and executed easily and subsequently reused for other data sets. Moreover, using RAPIDPROM, one can benefit from combinations of process mining with other types of analysis available through the RAPIDMINER marketplace.

1 Introduction

The number of events recorded in operational processes is growing every year. This applies to all domains: from healthcare and e-government to production and maintenance. Event data are a valuable source of information for organizations that need to meet requirements related to compliance, efficiency, and customer service. Within organizations the interest in data science and big data is rapidly growing. Therefore they need to handle and analyze novel sources of data in smarter and more efficient ways. However, in order to *improve business processes and services* it is often not sufficient to focus on data storage and data analysis alone. In fact, it is often the case that individual data elements within a process-based data set are not independent at all. *Process mining* [1] is a *process centric* technique that helps to turn event data into real value: by discovering the real processes, by automatically identifying bottlenecks, by analyzing deviations and sources of non-compliance, by revealing the actual behavior of people, etc.

Process mining is a rapidly growing sub discipline within both Business Process Management (BPM) [2] and data science [3]. Process mining is different from conventional data mining and machine learning techniques as it specifically takes into account that the event data originates from a business process. Many mainstream data mining and machine learning techniques on the other hand neglect this fact, i.e., they are *not* process centric. As a result, mainstream Business Intelligence (BI), data mining and machine learning tools are not tailored towards the analysis of event data and the improvement of processes. Fortunately, there are dedicated process mining tools able to transform event data into actionable process-related insights. For example, PROM [4] (www.promtools.org) is an open-source process mining tool supporting analyses such as process discovery, conformance checking, social network analysis, organizational mining, clustering, decision mining, prediction, and recommendation. Moreover, in recent years, several vendors released commercial process mining tools. Examples include: *Celonis Process Mining* by Celonis GmbH (www.celonis.de), *Disco* by Fluxicon (www.fluxicon.com), *Interstage Business Process Manager Analytics* by Fujitsu Ltd (www.fujitsu.com), *Minit* by Gradient ECM (www.minitlabs.com), *myInvenio* by Cognitive Technology (www.my-invenio.com), *Perceptive Process Mining* by Lexmark (www.lexmark.com), *QPR ProcessAnalyzer* by QPR (www.qpr.com), *Rialto Process* by Exeura (www.exeura.eu), *SNP Business Process Analysis* by SNP Schneider-Neureither & Partner AG (www.snp-bpa.com), and *PPM webMethods Process Performance Manager* by Software AG (www.softwareag.com). The growing number of process mining tools illustrates the relevance of process mining.

As mentioned above, PROM is a powerful open-source tool supporting hundreds of process mining analysis techniques. However, PROM does not support the creation and execution of *analytic workflows*. We therefore recently introduced RAPIDPROM [5]¹ (www.rapidprom.org), a RAPIDMINER extension that wraps around the core functionality present within PROM. It entails stable algorithms for the purpose of process mining analysis such as process discovery, conformance checking, performance analysis, etc. RAPIDPROM combines the advantages of the academic nature of PROM, i.e., it consists of state-of-the-art process mining algorithms, with the advanced data mining and analytic workflow capabilities of RAPIDMINER. Complex process mining workflows can be modeled and executed easily and subsequently reused for other data sets. Moreover, using RAPIDPROM, one can benefit from combinations of process mining with other types of analysis available through the RAPIDMINER marketplace.

In [1], three categories of process mining tools are identified, which are schematically depicted in Figure 1. *Type 1* process mining tools are mainly built for answering ad-hoc questions about business processes. An example tool of such type is *Disco*, which allows the user to interactively filter the data and project this immediately on a (newly learned) process model. Tools of *Type 3* are tailored towards answering predefined questions repeatedly in a known

¹RAPIDPROM is an open source project, the source code is openly available via <http://github.com/rapidprom>

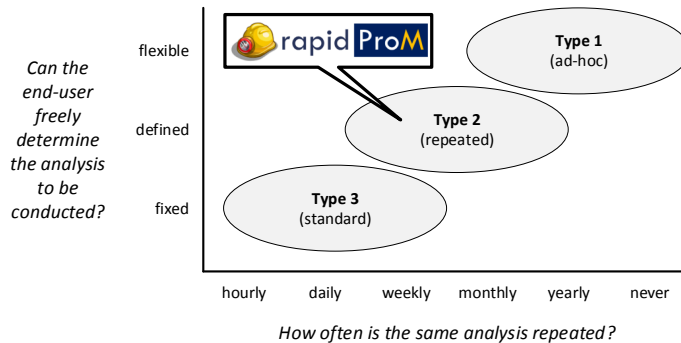


Figure 1: Diagram showing three different categories of process mining tools and the positioning of RAPIDPROM.

setting. These tools are typically used to create “process dashboards” that provide standard views. A tool like *Celonis Process Mining* supports the creation of such process-centric dashboards. Tools of *Type 2* aim to answer questions that are recurring, but possibly at a lower frequency. Analysis workflows may be predefined, but do not need to be completely fixed upfront. Customization may be needed and the interpretation of the results requires knowledge of process mining and understanding of the data. Unlike tools of *Type 1* and *Type 3*, the analytic workflow is made explicit such that it can be adapted and reused. RAPIDPROM is one of the few tools of *Type 2*. It can also be used as a *Type 1* tool, however, the workflow-like interaction style makes it predominantly a *Type 2* tool. RAPIDPROM can also be used for process mining research where experiments need to be repeated for different data sets and parameter values. Due to the *loop* and *subprocess* functionalities of RAPIDMINER, RAPIDPROM allows users to define a specific process mining analysis and repeat this multiple times while varying parameters or event data.

The development of the first version of RAPIDPROM started at the Eindhoven University of Technology around 2014 in context of the STW project “Developing Tools for Understanding Healthcare Processes” [5]. The extension consisted of basic operators for the purpose of process discovery, conformance checking, performance analysis, data exploration, etc. Since then, RAPIDPROM has been under active development, i.e., algorithms have been updated continuously, and, new operators were added. In [6], RAPIDPROM is used to assess the applicability of analytic workflows in a process mining context. Additionally the work offers a set of basic workflow patterns for the purpose of process mining.

The goal of this chapter is to provide a basic overview of process mining, and, in particular RAPIDPROM. We assume that the reader is not familiar with process mining, but has used RAPIDMINER before. The basic architecture is given and the most prominent objects and operators are discussed. We also present three case studies showing both the usefulness and applicability of applying

process mining by using analytic workflows.

The remainder of this chapter is organized as follows. In Section 2 we present a basic overview of the field of process mining. In Section 3 we briefly argue the main differences between conventional data mining and process mining. In Section 4 we present RAPIDPROM by means of an overview of its architecture, common objects and operators. In Section 5 we present three case studies highlighting different functionalities of RAPIDPROM. Finally, Section 6 concludes the chapter.

2 What is Process Mining?

The main goal of process mining is to improve operational processes by using event data. By exploiting recorded event data, process mining techniques are able to show *what actually* happened during the execution of a business process. We identify three main types of process mining, being *process discovery*, *conformance checking* and *performance analysis*. We present each of these types in more detail. Prior to this, we introduce the main source of data used within process mining, i.e., *event logs*. Throughout this section we use a simplified data set based on a real “Road Traffic Fine Management” process originating from an Italian municipality. In Section 5 we go more in depth and perform three case studies based on the real data set using RAPIDPROM.

2.1 Event Logs

Process mining is impossible without proper *event logs* [1]. Fortunately, event logs can be extracted from a wide variety of data sources, including enterprise systems (SAP, Oracle, etc.), hospital information systems (Epic, ChipSoft, Siemens, etc.), middleware (IBM, Microsoft, etc.), business process management systems, mobile applications, social media, sensor networks, etc. Most databases contain event information. However, often quite some efforts are needed to extract event logs from them (scoping, selection, and transformation).

An event log contains event data related to a particular process. Each *event* in an event log refers to an activity executed for a particular *process instance*, also referred to as a *case*. Events related to a case are ordered and can have any number of additional attributes. Examples of typical attributes next to the mandatory case identifier, activity and time attributes are resource, costs, transaction type, location, etc. Not all events need to have the same set of attributes. However, typically, events referring to the same activity have the same set of attributes.

A simplified example of an event log, based on a real event log, is depicted in Figure 2. The event log shows a few events related to handling traffic fines. Consider for example *row 2* in the example, i.e., the first event below the header. This event relates to a *Payment* activity, performed in context of case *S101157*. The *amount* of the fine equals the *amount of payment*, as indicated by the `amount` and `payment.amount` columns respectively. Note that additional infor-

| 1 | case | activity | time_stamp | amount | expense | payment_amount | article | vehicle_type |
|----|---------|--------------------------|-------------------------|--------|---------|----------------|---------|----------------|
| 2 | S101157 | Payment | 2005/03/02 00:00:00.000 | 35 | | 35 | 7 | FORD GALAXY |
| 3 | S101221 | Create Fine | 2005/03/02 00:00:00.000 | 35 | | | 158 | MITSUBISHI |
| 4 | S86430 | Create Fine | 2005/03/02 00:00:00.000 | 71 | | | 157 | FIAT UNO |
| 5 | S99595 | Create Fine | 2005/03/02 00:00:00.000 | 35 | | | 7 | FORD FIESTA |
| 6 | S99597 | Create Fine | 2005/03/02 00:00:00.000 | 35 | | | 158 | RENAULT TWINGO |
| 7 | C17393 | Create Fine | 2005/03/03 00:00:00.000 | 143 | | | 20 | |
| 8 | C17393 | Send Fine | 2005/03/03 00:00:00.000 | 143 | 0 | | 20 | |
| 9 | C17393 | Insert Fine Notification | 2005/03/03 00:00:00.000 | 143 | 0 | | 20 | |
| 10 | N69857 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | FIAT MAREA |
| 11 | N69858 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | RENAULT CLIO |
| 12 | N69859 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | TOYOTA |
| 13 | N69860 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | ALFA ROMEO |
| 14 | N69861 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | NISSAN MICRA |
| 15 | N69861 | Payment | 2005/03/03 00:00:00.000 | 35 | | 35 | 157 | NISSAN MICRA |
| 16 | N69862 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | FIAT PANDA |
| 17 | N70626 | Payment | 2005/03/03 00:00:00.000 | 35 | | 35 | 157 | FIAT PUNTO |
| 18 | N71668 | Payment | 2005/03/03 00:00:00.000 | 35 | | 35 | 157 | OPEL |
| 19 | N71670 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | NISSAN MICRA |
| 20 | N71670 | Payment | 2005/03/03 00:00:00.000 | 35 | | 35 | 157 | NISSAN MICRA |
| 21 | N71672 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | TOYOTA |
| 22 | N72127 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | OPEL CORSA |
| 23 | N72129 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | CHRYSLER |
| 24 | N72130 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 157 | IVECO |
| 25 | S101318 | Payment | 2005/03/03 00:00:00.000 | 35 | | 35 | 7 | PEUGEOT 205 |
| 26 | S101326 | Create Fine | 2005/03/03 00:00:00.000 | 35 | | | 7 | MERCEDES |

Figure 2: Simplified event log where each line refers to an *event* in the process of handling road fines.

mation such as the *law article* related to the fine as well as the *vehicle type* is also recorded. Figure 2 shows three events that are all related to case *C17393* (rows 7,8,9). Three activities are performed in sequence for this case, i.e., *Create Fine*, *Send Fine*, and, *Insert Fine Notification*. Together these three events form a *trace* describing the lifecycle of a traffic fine.

Figure 3 shows an overview of process mining highlighting the three main types of analysis. Event logs are represented by the three shapes labeled with *event data*. In the upcoming sections we discuss each sub discipline, i.e., process discovery, conformance checking, and, performance analysis in more detail. As the figure illustrates, all these techniques use, amongst other artifacts, event logs as an input.

2.2 Process Discovery

The first type of process mining is discovery (see Figure 3). A process discovery technique takes an event log as an input and produces a model without using any a-priori information. Typically the focus of process discovery techniques is on the *control-flow* aspect of a process. Within this aspect we are mainly interested in what ways the activities within the process can be ordered. Several process model formalisms exist, e.g., BPMN [7], Petri nets [8] etc.

A simplified example of a process model, using the Petri net formalism, is depicted in Figure 4. This process model was discovered based on the event log depicted in Figure 2. Within the model, a square with an inscription represents an activity. The squares without inscription refer to *invisible activities*. Such invisible activities are usually used for routing purposes, or, to allow us to skip certain activities, i.e., we are able to skip the *Payment* activity. This particular

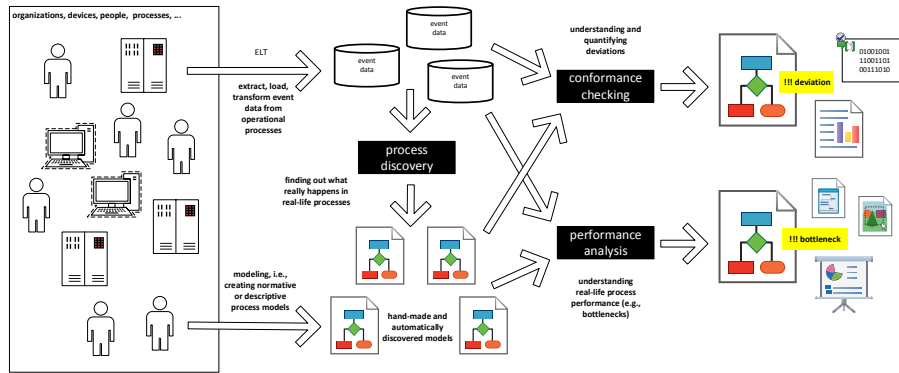


Figure 3: Schematic overview of process mining spectrum showing the three main types of process mining: (1) process discovery, (2) conformance checking, and (3) performance analysis.

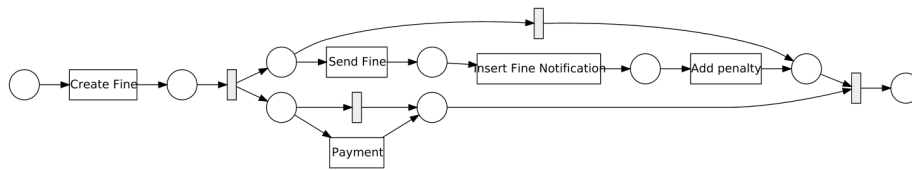


Figure 4: Simplified Petri net describing the main behavior present in the example event log of Figure 2.

process model describes that within the process, a fine is always created first. After this two *parallel branches of behavior* are started. The upper branch describes that the sequence of activities *Send Fine*, *Insert Fine Notification* and *Add penalty* should be performed, or, the activities are not performed at all. The lower branch specifies that a payment should be performed, or it should be skipped. If all activities within the two branches are performed or skipped, the process finishes. Note that the fact that the two branches are executed in parallel implies that the execution of the payment activity is independent of the execution of the activities in the upper branch. Thus, according to this process model, the payment activity can be performed at any point in time, though after the fine is created.

2.3 Conformance Checking

The second type of process mining is conformance checking (see Figure 3). Here, an existing process model is compared with an event log of the process that the model is describing, i.e., modeled behavior is confronted with observed behavior. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa.



Figure 5: An example conformance checking diagnostics for a particular process instance.

Consider again the simple process model depicted in Figure 4. As an example, we take the following actual trace from the event log used as a basis of the snapshot in Figure 2: $\langle \text{Create Fine}, \text{Send Fine}, \text{Insert Fine Notification}, \text{Add penalty}, \text{Send for Credit Collection} \rangle$. Obviously, this trace is not completely conforming with respect to the model as it contains activity *Send for Credit Collection*, which is not present in the model. Applying conformance checking techniques for this trace results in Figure 5.² The green blocks indicate that the algorithm was able to match an event in the trace to an activity in the model. The gray blocks correspond to the invisible activities as introduced in the previous section. Finally the yellow block corresponds to a non-conforming event. In this case, it refers to an activity that did occur in the trace of events, though the model does not describe the activity. Such type of non-conforming event is often referred to as a *log move*. Clearly, it is also possible that the model describes that a certain activity should happen, though the activity is not present in the trace of events. In such case we refer to a *model move*.

2.4 Performance Analysis

The third type of process mining is performance analysis by replaying the event data on a discovered process model (see Figure 3). Typically, techniques from conformance checking are used combined with timing information present in the event log, e.g., by using timestamps we are able to deduct the average time in-between two events etc. These types of statistics are subsequently projected onto a discovered or hand-made process model.

The results of performance analysis can be used to identify the problematic parts of the process, e.g., bottlenecks. An example of such analysis is depicted in Figure 6. Within the figure, we projected timing information from our example log on the process model of Figure 4. In this example, the yellow colored

²In Figure 5 we project conformance diagnostics onto a trace. However, conformance diagnostics can also be projected on the process model.

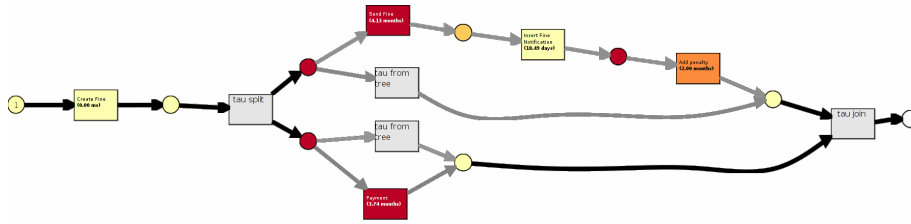


Figure 6: By replaying the event log on the process model, it is possible to annotate the model with performance information. The red transitions and places correspond to bottlenecks in the process.

elements of the process indicate efficient flows, whereas orange and/or red elements highlight activities that are taking longer. Again, the gray components correspond to invisible activities and are not considered within the performance analysis.

2.5 Other Types of Process Mining

Apart from the three subfields mentioned in the previous sections, other analyses have been developed within the domain of process mining as well. An example of analysis is predictive analysis, i.e., predicting the remaining time of a process instance. Also, some analysis methods focus on the organizational perspective of the process. Usually an event log contains information regarding what resource executed what activity. Hence, several interesting social networks can be mined from the event log. Consider for example Figure 7 which depicts a *similar task social network*.

The vertices within the social network correspond to resources that were active within the process. An arc between two vertices indicates that the resources execute similar tasks within the process. Clearly there are a few clusters identifiable within the network, suggesting that different groups of resources execute a different set of activities.

Another type of process mining is decision mining which focuses on the data flow in a process model. For each decision point, one can derive explanations based on different features. This can be used to learn that fines of a particular type, e.g., fines for speeding versus fines related to parking illegally, correlate with specific process executions.

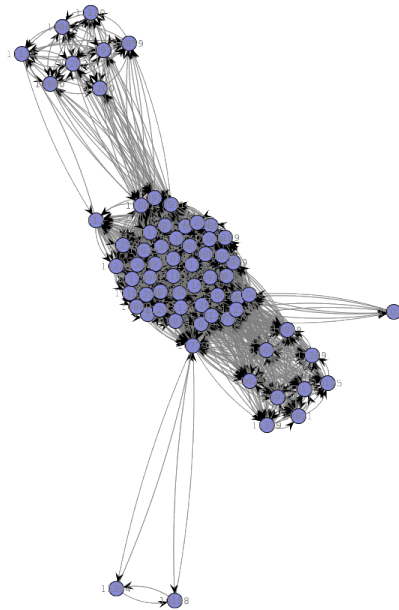


Figure 7: An example of a social network based on similarity of work profiles of resources.

3 Why is Process Mining Different From Data Mining?

Process mining provides a range of tools to improve processes in a variety of application domains. In the previous section we introduced the main forms of process mining. Process mining builds on process model-driven approaches and data mining. However, process mining is much more than an amalgamation of existing approaches. For example, existing data mining techniques are too data-centric to provide a comprehensive understanding of the end-to-end processes in an organization. Some would argue that process mining is part of the broader data mining or machine learning discipline. Depending on the definition, this could be (partially) correct. A common definition for data mining is “the analysis of (often large) data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner” [9]. Using this broad definition, parts of process mining are indeed included. However, discussions on such inclusion relations are seldom useful and are often politically motivated [1]. Most data mining tools do *not* provide process mining capabilities, most data mining books do *not* describe process mining techniques, and it seems that process mining techniques like conformance checking do *not* fit in any of the common definitions of data mining. It is comparable to claiming

that “data mining is part of statistics”. Taking the transitive closure of both statements, we would even be able to conclude that process mining is part of statistics. Obviously, this does not make any sense.

Like process mining, data mining is data-driven. However, unlike process mining, mainstream data mining techniques are typically not process-centric. Process models expressed in terms of Petri nets or BPMN diagrams cannot be discovered or analyzed in any way by the main data mining tools. Typically, data mining techniques assume that the data items used originate from some unknown distribution, and moreover, are *independent*. Within process mining, the primary source of data is an event log. At the lowest level, an event log consists of events, however, multiple events together constitute to a trace. Taking into account that events together constitute to cases effectively adds an additional layer within the data which can not be ignored. Thus, within process mining we do not only consider the events in isolation, rather, we also look at events at the trace level and use this to gain new insights within the data.

There are a few data mining techniques that come close to process mining [1]. Examples are sequence and episode mining [10, 11]. However, these techniques do not consider end-to-end processes.

One of the important features of RAPIDPROM is that the standard data mining techniques shipped with RAPIDMINER can be combined with a range of process-centric analytical techniques from PROM. Through process mining, it becomes easier to apply data mining techniques to event data. The process model serves as a backbone for a variety of data mining techniques (classification, clustering, etc.). For example, decision rules can be learned using standard data mining tools after the control-flow backbone (e.g., a Petri net) has been learned using a process mining tool. It is also interesting to combine process mining with other types of analysis available through the RAPIDMINER marketplace (e.g., text mining, web mining, deep learning, etc.).

4 RapidProM

In this section we describe RAPIDPROM, the process mining extension of RAPIDMINER. We present a high-level view of the basic architecture of the extension, commonly used objects, and, an overview of commonly used operators.

4.1 Architecture

As indicated in the introduction, the process mining toolkit PROM [1, 4] is the standard scientific tool for performing process mining analytics. As a consequence, PROM consists of a vast amount of state-of-the-art algorithms for the purpose of process discovery, conformance checking and performance analysis. Conveniently, both PROM and RAPIDMINER are programmed in the java (<http://www.java.com>) programming language. Hence, porting existing algorithms implemented in PROM is mainly concerned with integrating them within

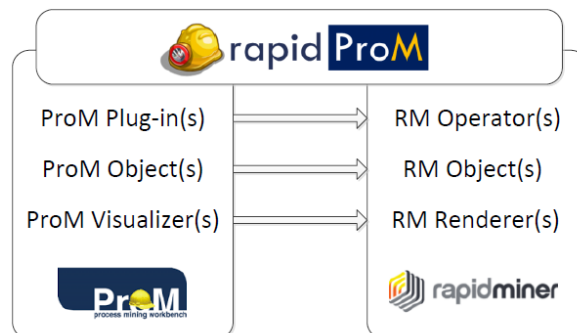


Figure 8: Basic overview of the architecture of RAPIDPROM.

the RAPIDMINER ecosystem.

In Figure 8 a high-level overview of the architecture of RAPIDPROM is depicted schematically. RAPIDPROM acts as a bridge between RAPIDMINER and PROM. Within PROM any algorithm that performs some process mining task, e.g., a process discovery algorithm is called a *plug-in*. Such plug-in usually expects a (set of) input object(s) and results in a (set of) output object(s). Thus in that respect a plug-in is comparable with an *operator* as defined within RAPIDMINER, and, most operators in RAPIDPROM correspond directly to their counterpart in PROM. Each plug-in has an associated (set of) visualizer(s) that allow the user to inspect the results of their process mining analysis. Again, most of these visualizers are ported as *renders* into RAPIDPROM and hence, have a corresponding counterpart in PROM.

4.2 Objects

Objects specific to RAPIDPROM are either based on data stored in some source, e.g., files, or, are the result of applying some algorithm. A complete overview of all objects ported and/or defined in RAPIDPROM is outside the scope of this chapter. However, we present the most prominent categories of objects relevant for process mining related analyses.

Event Logs. The most prominent data objects within RAPIDPROM are event logs. As indicated, an event log is a collection of traces which represent process instances that have been executed. Several operators within RAPIDPROM need an event log as an input object. RAPIDPROM supports importing data files which adhere to the XES standard (<http://www.xes-standard.org/>). Most commonly `.xes` files are used, an XML based data source adhering to the standard. Within RAPIDPROM, the OpenXES reference implementation is used for importing/exporting `.xes` files. Additionally, RAPIDPROM supports converting `example sets` into event log objects, and, event log objects into `example`

sets.³

Process Model Related Objects. Since most algorithms within process mining are related to (business) processes, several objects implementing some process model formalism are available in RAPIDPROM. Examples of such formalisms are BPMN Models [12], Petri nets [8], colored Petri nets [13], process trees [14], etc.

Analysis and Reporting Objects. Apart from event logs and process model related objects, RAPIDPROM includes several objects related to analysis and reporting. For example, we are able to perform conformance checking of event logs and Petri nets by computing so called *alignments* [15]. Hence, RAPIDPROM includes PROM based alignment objects with corresponding visualizations to inspect conformance results. Another example would be the *Inductive Visual Miner* [16] object, which presents a visual animation of cases going through a process model. This object is interactive, as it enables the user to filter cases based on selected behavior. It also highlights deviations of cases w.r.t. the process model.

4.3 Operators

RAPIDPROM introduces several operators that allow us to perform process mining analyses. We present the most prominent categories of process mining operators.

Input/Output. For some of the objects defined in RAPIDPROM *import* and *export* operators are available, e.g., Event logs, Petri nets, etc. All these operators need the input/output file/folder as a parameter. Additionally, for event logs an *extractor* operator is present which is able take a file as an input object.

Discovery. RAPIDPROM provides several different process discovery algorithms, e.g., the Alpha Miner [17], the Heuristics Miner [18], and the Inductive Miner [19]. Apart from these *control-flow* based algorithms, i.e. algorithms focusing on the sequential ordering of business activities within a process, RAPIDPROM also provides a Social Network Miner [20]. This operator allows us to discover a social network of interacting resources within the event log. The corresponding renderer allows the user to inspect and manipulate this network in an interactive fashion.

Analysis. RAPIDPROM provides several operators for the purpose of analyzing process discovery results. Several conformance checking related operators are present that allow the user to assess the conformance of an event log to a process model and vice versa. Furthermore, several operators that enable the user to analyze/enhance process models are provided, e.g., for analyzing formal properties of the models and/or simplifying the models. Finally, RAPIDPROM offers operators that allow the user to visualize actual data present in the event log onto the model, e.g., “replaying” the event log on a process model.

Apart from the three main categories mentioned here there are some more

³An `example set` is the basic tabular data object in RAPIDMINER.

operators available in RAPIDPROM. There are operators that provide the possibility to convert certain objects into other objects, e.g., Petri nets to BPMN models. Other operators allow the user to manipulate event logs, e.g., adding artificial start and end events to traces within event logs (which greatly improves the performance of some process discovery algorithms).

An example workflow containing an import operator, a discovery operator and an analysis operator is depicted in Figure 9.

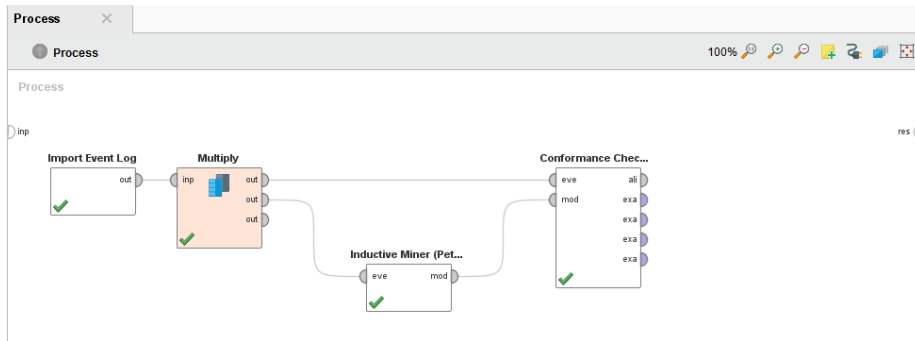


Figure 9: An example of a RAPIDPROM workflow where (i). an event log is imported, (ii). a process model is discovered based on the event log (using the Inductive Miner), and, (iii). a conformance checking analysis is performed using both the event log and the resulting process model.

5 RapidProM in Action

This section describes three concrete use cases in which the operators provided by RAPIDPROM are used to obtain insights about a process using the process mining techniques described in Section 2. We aim at using simple workflows in order to show the main *process mining capabilities* of RAPIDPROM. A real-life event log, containing more than 150.000 cases, is used as input for all three use cases. The event log is obtained from the data archive of the 4TU.Centre for Research Data (4TU.ResearchData).⁴ Within this archive, other event logs are publicly available as well, i.e., https://data.4tu.nl/repository/collection:event_logs as part of the “IEEE Task Force on Process Mining - Event Logs” collection. Within the case studies we do not go into detail w.r.t. the algorithmic properties of the operators used. However, the complete documentation (i.e., operator description, parameter explanation and related articles) of all the operators used within the workflows described in the case studies is provided by RAPIDPROM and is embedded in RAPIDMINER (by

⁴<http://data.4tu.nl>

means of the Help Window). Finally, note that some of the results depicted in this section are enhanced to increase their readability on paper and therefore in some cases might deviate from their corresponding visualization(s) in RAPIDPROM.

5.1 Dataset: Road Fines Management Process

The dataset used in the three case studies is an event log that was extracted from an information system that handles the “Road Traffic Fine Management” process in an Italian municipality.⁵ The event log includes cases of road traffic fines that are processed by the municipality over a period of three years (from January 2000 until June 2013). The “Road Traffic Fine Management” process is composed of 11 different activities. We describe the process as it *should* look like according to domain experts. The activities, as recorded within the event log, are written italic.

The process starts with a fine being created (i.e., *Create Fine*). After a fine has been created, it is sent to the offender’s place of residence (i.e., *Send Fine*). When the offender receives the fine, the date of reception of such notification is also registered (i.e., *Insert Fine Notification*). After this, the fine should be paid within 60 days (i.e., *Payment*). However, in case the fine was physically handed over to the offender, e.g. by means of a parking ticket, the offender is able to immediately pay the fine. In such case, the fine will not be sent and there will be no registration of the notification. This exception, i.e., direct payment after ticket creation saves the offender administration costs. In total, the offender has 60 days to either pay the fine or appeal against it. After this period, a penalty is added to the fine amount (i.e., *Add Penalty*). If the offender appealed against the fine within 60 days, the appeal is sent to the corresponding prefecture (i.e., *Send Appeal to Prefecture*), which is registered when it is received (i.e., *Insert Date Appeal to Prefecture*). The results of the appeal are sent back to the municipality (i.e., *Receive Result Appeal from Prefecture*) and they are notified to the offender (i.e., *Notify Result Appeal to Offender*), which can appeal against the result (*Appeal to Judge*). If the offender does not pay (possibly after a denied appeal), the fine is sent for credit collection (i.e., *Send for Credit Collection*).

A simple normative process model (Petri net) based on the description in the previous paragraph is depicted in Figure 10. Note that this model does not capture any form of parallelism. It is merely a sequence of activities combined with possible choices, e.g., after the *Create Fine* activity, the process model describes the choice of either a *Payment* activity, or, a *Send Fine* activity.

5.2 Case Study 1: Discovering Process Models

As mentioned in Subsection 2.2, many different models can be discovered from the same event log by applying different discovery techniques. Furthermore, process models can be described using different notations, e.g., using Petri net,

⁵This event log is publicly and permanently available at (<http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>).

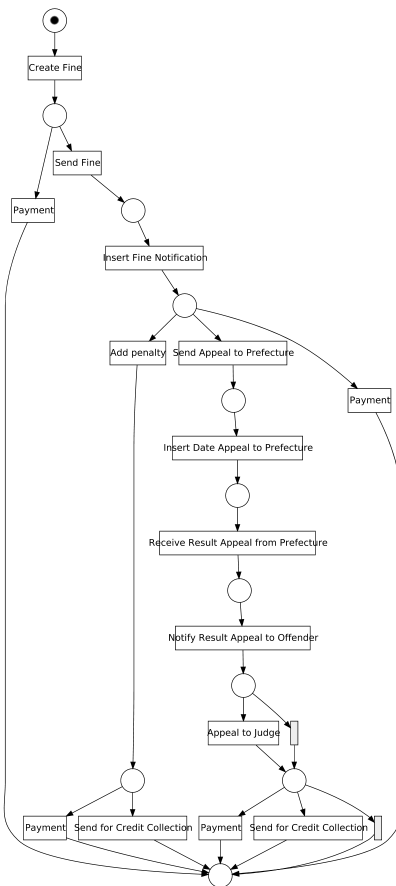


Figure 10: Normative process model based on the “Road Traffic Fine Management” process description.

BPMN, etc. This case study shows how several process models can be discovered from a real event log using the *discovery* operators of RAPIDPROM.

Workflow Figure 11 illustrates a basic RAPIDMINER workflow used to discover process models from an event log. First, an event log is imported using the *Import Event Log* operator. Then, several process models are discovered from the event log using different operators that produce process models in different notations. The Discovery operators included in this workflow, and the type of process model that they produce are: the *Alpha Miner* (Petri net), the *ILP Miner* (Petri net), the *Heuristics Miner* (heuristics net), the *Inductive Miner* (Petri net), the *Social Network Miner* (social network), the *Transition System Miner* (transition system), and the *Fuzzy Miner* (fuzzy model).

The workflow immediately highlights an advantage of using RAPIDMINER

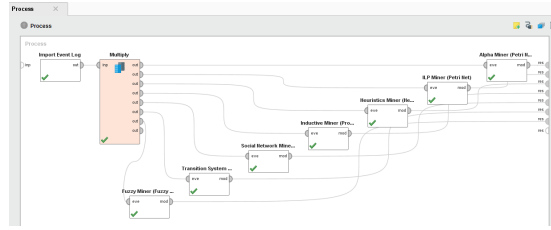


Figure 11: Workflow used for Case Study 1: an event log is imported and several process model are created using some of the available discovery operators provided by RAPIDPROM.

as it allows us to generate all results of the different miners in one go. Moreover we are able to directly reuse this workflow for other event logs as well. In such case we just change the file pointed to by the *Import Event Log* operator. If we do this, depending on the event log, we might need to change the *classifier* used by the discovery algorithms. The classifier parameter is shared by all discovery operators. It is a default attribute within an event log and specifies what *event attribute* should be used in order to identify the corresponding *activity*. In case of our example event log snapshot, depicted in Figure 2, the classifier to use is the **activity** column.⁶ Finally, the workflow can also be used as a sub-process after which the best model, depending on some process model specific quality criteria, e.g., model simplicity, is selected for further analysis.

Results Analysis Consider Figure 12, in which the result of applying the Heuristics Miner on the event log is depicted. The model depicted in Figure 12 is a Heuristics net. The squares refer to activities whereas an arrow between two activities indicates that the source activity preceded the target activity. The intensity of the different elements within the net correspond to their relative

⁶Classifiers originate from the XES standard. More information on classifiers can be found in: http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf.

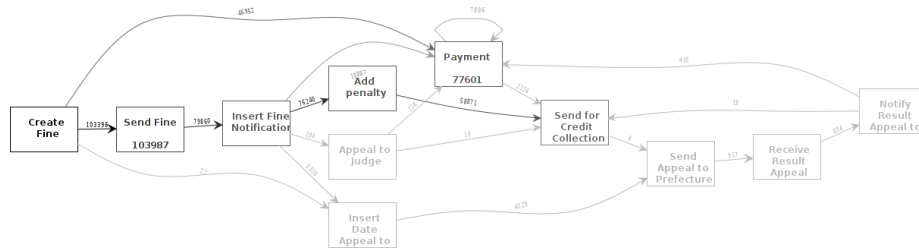


Figure 12: Result (heuristics net) of applying the Heuristics Miner on the event log.

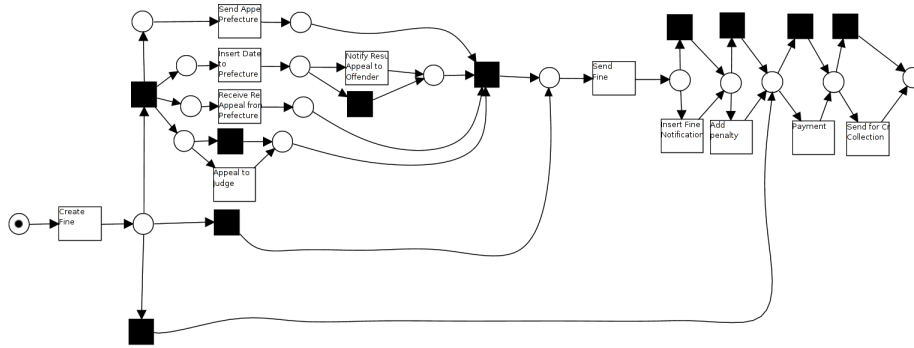


Figure 13: Result (Petri net) of applying the Inductive Miner on the event log.

frequency as observed within the event log. Although the presence of parallelism is somewhat hard to detect in a heuristic net, it still provides very usable insights. From the heuristics net we deduce that indeed, as prescribed in the textual and normative models, after the *Create Fine* activity a *Payment* activity can be performed. However, when considering the *Payment* activity, we observe a *self-loop*, i.e. an arc leaving and entering the *Payment* activity. This possibly suggests that the specific municipality allows for paying the fine in terms. Another interesting observation is the fact that after a penalty is added, the activity *Send for Credit Collection* always seems to occur. In some cases, even though a payment was made, still the *Send for Credit Collection* activity was performed. This can have several explanations, e.g., the payment did not match the fine, the payment was registered too late etc. Finally after the *Insert Fine Notification* activity, the *Appeal to judge* activity seems to be executed.

Consider the Petri net depicted in Figure 13, obtained by applying the Inductive Miner on the event log (using a noise filter level of 0.2). Due to the application of a noise filter, the Petri net might neglect some of the behavior as captured within the event log. The model describes that after the *Create Fine* activity we are able to choose between executing three invisible activities (the black squares, intended for routing purposes). Choosing the lowest invisible activity leads us directly to the *Payment* activity. We are however able to skip the activity. If we execute the middle invisible activity (after the *Create Fine* activity), we always have to execute the *Send Fine* activity. After this the *Insert Fine Notification* activity is either executed or skipped. Finally, if we choose the upper invisible transition, the model shows some interesting behavior. The activities *Send Appeal to Prefecture*, *Insert Date to Prefecture* and *Receive Result Appeal from Prefecture* should all be executed. However, the model describes that they are in a parallel block, i.e., we are able to execute them in any arbitrary order. Note that the model also describes that we are able to only execute the *Create Fine* activity and then skipping all subsequent activities, i.e., resulting in a trace with only one activity.

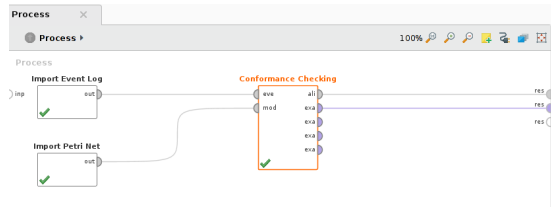


Figure 14: Workflow used for Case Study 2: an event log and Petri net are imported. Subsequently the model and the event log are used for conformance checking.

5.3 Case Study 2: Checking Process Conformance

The previous case study shows how process discovery can be used to get an insight in a process based on the *actual behavior* recorded in an event log. Very often, as was the case in the previous case study, the event log contains behavior that does not comply with the process description. This may have several reasons, e.g., the process description is incomplete, errors occurred during logging the process, the process was executed incorrectly etc. At the same time, due to limitations of the process discovery algorithm, a resulting process model may describe behavior that is *not present in the event log*. Although applying process discovery yields insights w.r.t. the degree of compliance, it does not provide an accurate exact result. Therefore, in this case study, we perform conformance checking of the normative process model w.r.t. the event log to get a more exact conformance result.

Workflow Figure 14 illustrates the basic workflow for the purpose of conformance checking. We import an event log using the *Import Event Log* operator. Secondly we import the normative model using the *Import Petri net* operator. The two artifacts are used to check conformance by means of the *Conformance Checking* operator. Note that the *Conformance Checking* operator needs a Petri net and an event log as an input, hence, the Petri net can also be the result of any process discovery operator (given that the operator produces a Petri net). Hence, we can use the result of a conformance checking operator as a quality measure for a process discovery algorithm. The *Conformance Checking* operator provides several results, e.g., a projection of conformance results onto the process model, or, onto the event log. Also, conformance checking results are provided as **example sets** for further analysis using other RAPIDMINER operators.

Results Analysis In this section we analyze some of the results obtained by the *Conformance Checker* in the workflow described in the previous section. We first inspect the projection of the conformance checking results onto the process model, as depicted in Figure 15. Note that the model depicted in Figure 15 is the same as the normative model depicted in Figure 10, the difference in layout is related to the underlying visualization software. Within Figure 15 we identify two different types of squares, which we highlight in Figure 16.

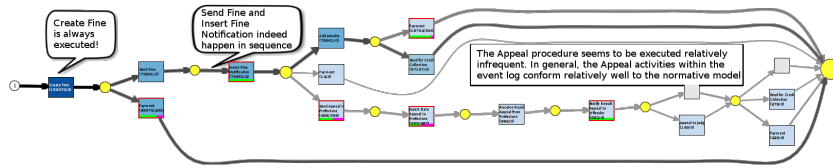


Figure 15: Conformance results projected onto the normative model

The first type of activity are highlighted in 16a, are highlighted in a solid blue color. For these type of activities, the event log and the model are perfectly conforming w.r.t. each other. The intensity of the color is related to the frequency of occurrence of such an activity. As highlighted in Figure 15, the *Create Fine* activity of the normative model aligns perfectly w.r.t. the event log (and vice versa). This mainly implies that the activity is executed in each trace. It does however not necessarily imply that the *Create Fine* activity is indeed always the first activity within any sequence. The exact reason for this is somewhat technical and outside of the scope of this chapter, hence, we refer the interested reader to [15] for more detail. The *Create Fine* activity is also depicted in 16a. The (150370/0) inscription underneath the *Create Fine* label indicates that in 150.370 cases the event log and the model agreed that the activity should be executed, whereas in 0 cases this was not the case. Another example of a perfectly conforming activity is the *Send Fine* activity.

The second type of activity are activities that are not perfectly conforming. These activities have a red border, and, a green and pink bar in the bottom of the square. Consider Figure 16b for an example. The width of the green bar indicates the number of conforming occurrences of the particular activity. The pink bar indicates the number of non conforming occurrences. In case of the *Payment* activity, depicted in Figure 16b, we observe that in 49.976 cases the event log and model conform w.r.t. executing the activity. However, in 20.534 cases, the model dictates that the activity should occur, whereas according to the event log, this is not the case.

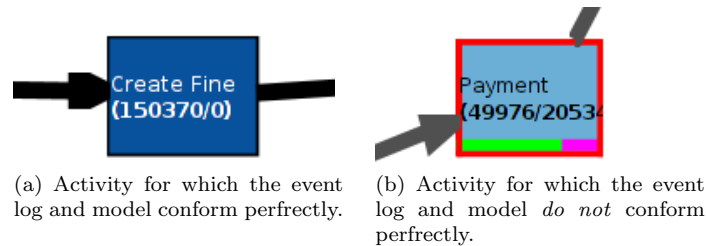


Figure 16: Two activities from the normative model having a different type of conformance result.

Although the projection of conformance metrics onto the model leads to interesting insights, it does not allow us to inspect the conformance of individual traces within the event log. Moreover, as argued before, we are not able to deduce that the *Create Fine* activity is indeed always the first activity of any case. Within RAPIDPROM, we therefore also provide the option to project the conformance results onto case within the event log (by means of a different *renderer* of the conformance results). An example screen shot of this view in RAPIDPROM is depicted in Figure 17. Within this view, the cases are ordered



Figure 17: Example screen shot of conformance statistics projected onto cases in the event log.

based on their frequency within the event log. Consider Figure 18 depicting the first case of Figure 17 in more detail. As the statistics show, there is a total of 56.482 traces within the event log that exactly follow this execution path. Clearly this is a typical case of non-paying offenders that did not appeal.

There are also cases for which the trace and the model are not conforming w.r.t. each other. Consider the example depicted in Figure 19. The figure indicates that indeed, the *Create Fine* activity is both present in the trace and in the model. Secondly, it indicates that a *Send Fine* activity was performed, followed by an *Insert Fine Notification* activity. Somehow, the offender did not pay in time and hence a penalty was added to the fine. Up until this point the trace and the model still conform. However, after the first *Payment* activity, the offender again performed a *Payment* activity. The second *Payment* is not in line with the normative model and is therefore indicated in yellow. There may be several reasons for a duplicate payment, e.g., the offender paid the ticket



Figure 18: Conformance results projected onto a trace which perfectly conforming w.r.t. normative model.



Figure 19: Conformance results projected onto a trace which does not conform w.r.t. the normative model.

too late and afterwards payed the fine as well, the offender payed in multiple terms etc. In any case, given that this behavior occurred around 3.736 times, such case is interesting to discuss with the business owner of the process, the Italian municipality in this case.

5.4 Case Study 3: Identifying Bottlenecks in a Process

The previous case study shows how to do conformance checking using an event log and a process model. In this case study, we go one step further and analyze the behavior in the event log from a *performance* perspective, as described in Subsection 2.4.

Workflow The workflow used within this case study equals the workflow used within the previous case study. However, instead of using the *Conformance Checking* operator, we use the *Analyze Performance (Manifest)* operator. The result of this operator is a projection of performance statistics onto the input model, in this case the normative model.

Results Analysis Some global statistics related to the performance of the process are depicted in Figure 20 The average throughput time reported is 10.51 months, with a standard deviation of around 11.45 months. Interestingly the longest running case was running for 114.57 months. The reason for this can be a long running appeal against the fine. However, often these type of cases are related to issues in data quality, e.g. inaccurate logging of events etc. Thus, this stresses the need for proper data cleaning and filtering methods.

To gain more insights into the performance of individual activities we show the result of projecting performance information onto the normative process model in Figure 21. The timing information in the event log is projected onto

| Case Property | Value |
|-------------------------------|---------------|
| #Cases | 150370 |
| #Perfectly-fitting cases | 112373 |
| #Non-fitting cases | 37997 |
| #Properly started cases | 150370 |
| Case Throughput time (avg) | 10.51 months |
| Case Throughput time (min) | 0.00 ms |
| Case Throughput time (max) | 114.57 months |
| Case Throughput time (std...) | 11.45 months |
| Observation period | 163.90 months |

Figure 20: Screens shot of the global performance statistics of the normative process as reported by RAPIDPROM.

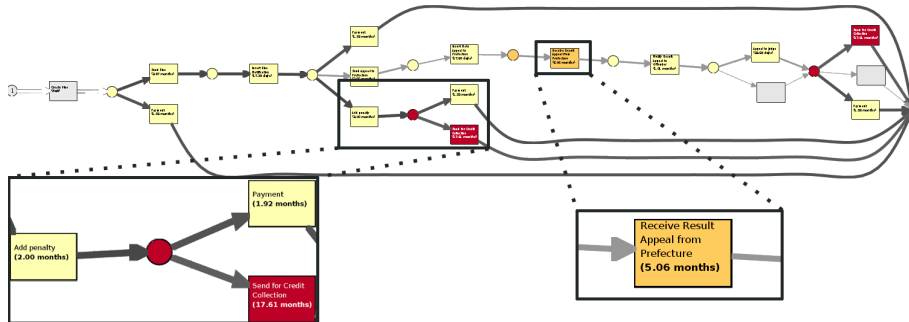


Figure 21: Projection of performance information into the normative process model.

the activities present in the model. The activities in the process model are colored depending on their performance, i.e., red colors indicate longer waiting times and yellow colors indicate shorter waiting times.

The result object is highly interactive: users can filter cases based on their throughput time, choose a different performance annotation e.g., sojourn time, waiting time, change color schemes, among many other functionalities. Also, for each activity several statistics are available such as minimum time, maximum time, mean, standard deviation etc.

The activity with the highest waiting time is *Send for Credit Collection*, having a waiting time of 17.61 months on average. This high waiting time can be explained by the fact that only those fines that are not paid are sent to credit collection. Also, the time that the municipality waits before sending an unpaid fine for credit collection depends on the actions of the offender (e.g., if he/she appealed against the fine).

The second highest waiting time corresponds to the activity *Receive Result Appeal from Prefecture*, with 5.06 months on average. It seems that a large amount of time is spent between applying for an appeal and receiving the result from Prefecture. Whether the municipality is able to speed up this part of the process is questionable, since this is most probably executed by an external party, i.e., the prefecture.

An interesting observation is the fact that activity *Add Penalty* has a waiting time of 2.0 months on average. This is in line with the fact that the municipality adds a penalty 60 days after the fine was sent. The standard deviation for this particular activity is around 30 minutes. Due to this small deviation it is highly likely that adding the penalty is an *automated activity*. Another interesting observation is related to payments. The average waiting time is 1.92 months, just under 2 months, i.e., the regular term for paying a fine. Although this seems a reasonable period, the standard deviation is 4.08 months, indicating that the average wait time for payments is not strictly related to the legal payment term.

As the previous insights highlight, performing performance analysis using RAPIDPROM leads to interesting results. These results can subsequently be used to select and filter the event log in order to inspect the bottleneck cases in more detail.

6 Conclusion

In this chapter we introduced the RAPIDPROM plugin which extends RAPIDMINER with process mining capabilities. RAPIDPROM offers support to organizations that need to manage non-trivial operational processes. Most of the operators provided by RAPIDMINER correspond to data handling and classical forms of data mining. These operators are *not* process-centric and cannot be used to analyze and improve end-to-end processes. The operators in RAPIDPROM provide *process centric* analysis capabilities. The RAPIDPROM operators focus on the analysis of event data and process models. They take into account that events are related to process instances and should be handled as such. RAPIDPROM supports discovering process models from event logs, checking conformance of an event log w.r.t. a process model, and calculating performance results based on an event log and a process model. Combined with the powerful capabilities of RAPIDMINER in terms of building analytical workflows and data mining, RAPIDPROM enables us to:

1. Design complex process mining experiments, combining several different techniques.
2. Reuse earlier defined analyses to be applied to new event logs (of the same or another process).
3. Bridge the gap between process mining and data mining.

We aim at keeping RAPIDPROM up-to-date as new process mining techniques or more efficient implementations emerge. For user/developer documentation, data sets, example workflows, etc., we refer to www.rapidprom.org.

Acknowledgements

The authors would like to thank all that contributed the PROM plug-ins used in RAPIDPROM (see www.promtools.org). Special thanks go to Ronny Mans who developed the first version of RAPIDPROM.

References

- [1] W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, Berlin, 2016.

- [2] W.M.P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, pages 1–37, 2013. doi:10.1155/2013/507984.
- [3] W.M.P. van der Aalst. Data Scientist: The Engineer of the Future. In K. Mertins, F. Benaben, R. Poler, and J. Bourrieres, editors, *Proceedings of the I-ESA Conference*, volume 7 of *Enterprise Interoperability*, pages 13–28. Springer-Verlag, Berlin, 2014.
- [4] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, A. Rozinat, H.M.W. Verbeek, and T. Weijters. ProM: The Process Mining Toolkit. In *Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009)*, 2009.
- [5] R. Mans, W.M.P. van der Aalst, and E. Verbeek. Supporting Process Mining Workflows with RapidProM. In L. Limonad and B. Weber, editors, *Business Process Management Demo Sessions (BPMD 2014)*, volume 1295 of *CEUR Workshop Proceedings*, pages 56–60. CEUR-WS.org, 2014.
- [6] A. Bolt, M. de Leoni, and W.M.P. van der Aalst. Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation. *International Journal on Software Tools for Technology Transfer*, pages 1–22, 2016.
- [7] OMG. Business Process Model and Notation (BPMN). Object Management Group, dtc/2010-06-05, 2010.
- [8] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
- [9] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT press, Cambridge, MA, 2001.
- [10] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14. IEEE Computer Society, 1995.
- [11] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [12] OMG. Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03, 2011.
- [13] K. Jensen and L.M. Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.
- [14] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Scalable Process Discovery with Guarantees. In K. Gaaloul, R. Schmidt, S. Nurcan,

- S. Guerreiro, and Q. Ma, editors, *Enterprise, Business-Process and Information Systems Modeling (BPMDS 2015)*, volume 214 of *Lecture Notes in Business Information Processing*, pages 85–101. Springer-Verlag, Berlin, 2015.
- [15] W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
- [16] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Process and Deviation Exploration with Inductive Visual Miner. In L. Limonad and B. Weber, editors, *Business Process Management Demo Sessions (BPMD 2014)*, volume 1295 of *CEUR Workshop Proceedings*, pages 46–50. CEUR-WS.org, 2014.
- [17] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
- [18] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven, 2006.
- [19] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In J.M. Colom and J. Desel, editors, *Applications and Theory of Petri Nets 2013*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer-Verlag, Berlin, 2013.
- [20] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative work*, 14(6):549–593, 2005.