Guided bottleneck identification in business process event logs

Copier, M.

Award date:
2020

Department of Mathematics and Computer Science
Analytics for Information Systems Research Group

Guided bottleneck identification in business process event logs

Master Thesis

Martijn Copier

Supervisors:

dr. ir. Dirk Fahland
dr. ir. Roeland Scheepens

Committee:

dr. ir. Dirk Fahland
dr. ir. Roeland Scheepens
dr. ir. Remco Dijkman

Eindhoven, January 2020

Abstract

Businesses operate by executing numerous business processes. Business Process Management (BPM) is the discipline of attempting to manage and improve business processes. As an extension to BPM, process mining is a relatively young research discipline which develops techniques for analyzing business processes. Performance analysis of business processes is a major aspect of process mining. However, recent research has shown that current process mining techniques only give limited insight into the performance analysis aspect of business processes, since the techniques do not capture *variability* in performance data. In this thesis, we propose and develop an approach for extracting, analyzing and visualizing variability in performance data. The method is evaluated using comparisons to contemporary analysis, and validation tests with domain experts. The comparisons indicated that our approach is accurate. Additionally, the domain experts noted that the approach assisted them with identifying and analyzing bottlenecks, in a way that was not possible before.

Contents

Chapter 1

Introduction

Current businesses operate by executing numerous activities in a structured way. These activities are performed by people or equipment, which, when performed in a certain sequence, create a product or service. The set of activities performed to create this product or service is called a *business process*. Business Processes can be administrative processes, such as the onboarding of a new employee. Procure-to-pay processes, such as purchasing, invoice management and paying, or information technology processes, such as ticketing system processes. Business processes are most commonly described using business process models. These models indicate which activities have to be executed, who should execute these activities, when, and in what order they should be executed. The act of monitoring and improving business processes is generally referred to as Business Process Management (BPM). Companies strive to improve business processes to save costs, improve throughput times, and improve product/service quality. Several sets of tools and techniques have been created to aid in improving business processes. Process mining is one such set of tools and techniques.

This thesis relates to business process management, process mining, and process bottlenecks. These concepts are explained in more detail in the coming sections. Additionally, we describe the research methodology used, and describe the company at which the research is performed (ProcesGold). Finally, we provide a section describing the structure of this thesis.

1.1 Business process Management

Business process management (BPM) is the discipline of attempting to improve business processes [1]. This is done by discovering, modeling, measuring, analyzing, optimizing, and automating existing business processes. A company benefits from improving their business processes, since it can lead to lower production costs, improved throughput time, or improved product quality [2]. To keep track of all business process activities performed within a company, *process-aware information systems (PAIS)* are used to record process information. This has become a standardized business practice in the past decade [3]. Examples of these *PAIS* are Enterprise Resource Planning (ERP) systems, Workflow Management (WFM) systems, Business Process Management (BPM) systems, etc.

Business processes are not static. While these processes can be defined beforehand, external factors cause the processes to not behave as desired, creating a need for adapting them. Factors, such as, new emerging markets, changes in governmental laws and emerging/disruptive technologies can cause changes in the environment of these processes. Many companies have to navigate these changing environments and adapt their processes accordingly to keep a competitive advantage [4]. Whenever a business processes does not perform as desired (as determined by *Key Performance Indicators* (KPIs), or other measures), there is a need to address this problem. Problematic

business processes should be analyzed and (if needed) redesigned. A relatively young research discipline called *Process Mining* has gained popularity over the past decade. Process mining is the discipline of extraction, analysis, and visualization of the information produced and stored by *PAIS*.

1.2 Process Mining

The systems described in Section 1.1 save process related information in so-called *event logs* [5]. Such an event log consists of *events*, which contain information regarding: which activity was performed, what work item (loan applications, fines, subsidy requests, etc.) this activity was performed upon, and the time at which this activity was performed. Additionally, *events* may contain information about which resource executed the event, and other such attributes.

Events are grouped into *cases*. Depending on the type of work a company performs, *cases* are sequential records of events that follow a work item throughout its lifetime, identified by a Case ID. These events can be performed automatically (by a machine) or manually (by a human).

Real-life event logs can contain hundreds if not thousands of cases. To get a better grasp on all these cases, and the deeper analysis of event logs, the discipline of *Process Mining* has been researched and developed for the past decade. It is being applied to process data of corporations and (semi-)governmental sectors more frequently [2]. The goal of process mining is to extract process-related information from *event logs*. Process Mining contains several techniques to extract useful information from *event logs*. This information is used for several different purposes [6]. Firstly, *process discovery* is the technique of extracting, analysing, and modelling process based on the information contained in an event log. Secondly, *process conformance checking* is a technique which takes an event log of a process, and compares this to a predefined *process model* of the same process. This is to check whether the behaviour of a business process is reflected by the model it is based on. Lastly, *process enhancement* is used to extend or improve an existing business process using information gained from the analysis of the event log.

Several perspectives are considered within process mining [7]. The *control-flow perspective* focuses on the control-flow, i.e., the ordering of activities. The *organizational perspective* focuses on information about resources in the log, i.e., which resources are used, when, and for what? The *case perspective* focuses on properties of cases, i.e., what set of events and attributes are part of each case. Finally, the *time perspective* is concerned with the timing and frequency of events. This information can be used to discover performance bottlenecks and improve throughput times.

1.3 Process bottlenecks

Business processes do not always perform as planned, and often have so-called *bottlenecks*. In production, a bottleneck can be described as a part of the chain of processes, which for some reason, limits the flow of the entire chain. This can result in lower production rates, supply overstock, pressure from customers, and low employee morale.

From a business process perspective, the term "You're only as fast as your slowest team member" comes to mind. Here, it refers to (a set of) activities that negatively influence the throughput/waiting/process times of the entire process. This negative influence can be caused by many things: (unforeseen) bad design of the process model, lacking resources, oversaturation of the process, inefficient/unused steps with the process model, and more. For a business, these bottlenecks can ultimately lead to delays, losses in revenue, and customer dissatisfaction. For the remainder of this thesis, when the term *bottleneck* is used, it is defined as follows: A *bottleneck* is (a set of) activities that negatively influence the throughput times of the entire process. Negative influence

can be defined as having a negative influence on any KPI a company deems important, such as, maximum throughput time, minimal resource usage, revenue growth, support resolution time etc.

1.4 Research motivation and question

The *time perspective* is a major aspect of process mining. As stated in *Process Mining: Data science in action*, "The presence of timestamps enables the discovery of bottlenecks, the analysis of service levels, the monitoring of resource utilization, and the prediction of remaining processing times of running case" [7]. However, for this analysis to be accurate and functional, there is a need for *precise* knowledge about process behaviour and performance [8]. Models have been created for predicting thoughput and cycle times [9, 10]. Generally, the nodes and edges of these models are annotated with performance measures (i.e., average throughput time, waiting time, etc.). These performance measures are based on the aggregation of (historic) event timing information. While aggregation of performance data is widely used within the process mining discipline, such aggregations are based on the assumption that performance of a case does not change over time, and is not influenced by other cases.

Research done in the paper *Unbiased, Fine-Grained Description of Processes Performance from Event Data* [11] has shown that such assumptions are often made by a lack of a more precise understanding of the (changes in) process performance across cases and over time. In this paper, a novel visualization for process performance data is proposed. Additionally, a taxonomy of performance phenomena is created in the form of elementary and composite performance patterns. Validation of several real-life event logs confirms that process performance is neither stationary nor that cases are isolated from each other.

The PSM has show that *variability in performance* exists in event logs. However, there has not been any research on developing methods to (automatically) discover, analyze, and visualize this variability. We hypothesize that developing such methods will lead to improvements within business process management, and can be used as an additional tool within process mining. Therefore, we try to answer the following research question:

RQ: Can variability in performance be automatically discovered, analyzed, and visualized to facilitate bottleneck identification?

This research question breaks down into several secondary research questions. In Chapter 3, we explore the general research question in detail, and developed the following set of secondary research questions:

RQ 1: Can performance information, extracted from event log segments, be (automatically) subdivided into different performance classes which each represent a part of the variability in performance?

RQ 2: Can a set of analyses be created, as to discover similarities/differences within and between the performance classes?

RQ 3: Can the outcome of the performance class analysis be aggregated and visualized in such a way, as to support bottleneck identification?

First, in Chapter 2, we explain all relevant background knowledge needed to understand this thesis. In Chapter 3, we first explore limitations of the current approach to *timing analysis*, and explain how this approach can lead to misinterpreting performance information. We explore literature which highlights this limitation/misinterpretation. Finally, we elaborate upon the main and secondary research questions. In Chapter 4, we explore several methods for answering the research questions. Additionally, we present which methods are utilized for answering them. In

Chapter 5, we describe the required input format, how performance information (contained in *segments*) can be extracted from event logs and how the cases in these segments are classified into distinct groups automatically. Furthermore, this chapter aims to answer **RQ 1**. In Chapter 6, we explain several measures which quantify characteristics of the segment groups. We describe how these characteristics may indicate bottlenecks, and we describe a ranking which ranks the segments on their likelihood of containing a bottleneck. Furthermore, this chapter aims to answer **RQ 2**. In Chapter 7, we describe the visualization we created to aid in bottleneck identification using the created ranking. We explain how each of the four *perspectives of process mining* are represented within our visualizations. Furthermore, this chapter aims to answer **RQ 3**. In Chapter 8, we evaluate the correctness of our work, by comparing our findings to BPIC reports. Additionally we evaluate the understandability and usefulness of our work, by performing validation tests with process analysts from ProcessGold (explained in Section 1.5). Finally, in Chapter 9, we discuss the conclusions, limitations and future work of this thesis.

1.5 ProcessGold platform

This thesis is performed at ProcessGold, which is a software company based in the Netherlands. It currently develops one of the leading process mining platforms. This platform is able to analyze event data and is used to create company-specific process mining analysis dashboards. Functionalities such as process-flow comparison, conformance checking, variant overviews, frequency and timing information, resource utilization, and various data visualization tools are available within the platform. Findings of this thesis are visualized within the ProcessGold platform. These visualizations are further elaborated upon in Chapter 7.

Chapter 2

Preliminaries

This chapter aims to describe the background knowledge needed to understand this thesis. In Section 2.1, we explain what *event logs* are, what their structure is, and how *bottleneck* information is caught within event logs. In Section 2.2, we explain how the *timing perspective* in process mining works with a practical example. In Section 2.3 we explain how *segments* are defined in the context of this thesis, how segments are detected in event logs, and how performance information is extracted from these segments. In Section 2.4, we explain recent research done on *frequency analysis*, which provides a statistical basis for attribute analysis. Finally, In Section 2.5, we describe several template dashboard available in the ProcessGold Platform, and link these the process mining perspectives.

2.1 Event logs

2.1.1 Structure

Event logs consists of a combination of the 3 parts described below:

- **Case ID**, an identifier that defines that a particular case
- **Activity**, label of the activity that is performed
- **Timestamp**, time when the *event* occurred

Depending on the type of work a company performs, *cases* are sequential records of events that follow a work item (loan applications, fines, subsidy requests, etc.) throughout its lifetime, identified by a Case_ID. These events can be performed automatically, by a machine, or manually, by a human. As described in Section 1.1, anytime an action is performed, the BPM system saves a record of this action in the form of an *event*. Any set of such events can be combined into an *event log*. In practice, *event logs* can contain thousands to millions of events. Table 2.1 shows a small snippet of an event log of a synthetic invoice payment process.

2.1.2 How do event logs capture possible bottleneck information?

As discussed in Section 1.3, bottlenecks are (parts of) activities that negatively influence the throughput/waiting/process times of the entire process. Since event logs hold a record of what activities are performed when, and which activities pre/pro-ceed them (which is inferred from their timestamp), the event log also implicitly holds information on possible bottlenecks. These bottlenecks can be discovered because it is not only possible to determine the time spent on an activity, but it is also possible to compare different executions of the same activity, and see whether they have deviating (performance) characteristics.

Case ID	Properties	
	Activity	Timestamp
1	Receive invoice	01/01/2018 10:00:00
	Check received invoice	01/01/2018 10:00:02
	Final check of invoice	03/01/2018 13:00:12
	Approve Invoice	05/01/2018 09:35:15
	Pay Invoice	06/01/2018 16:25:54
2	Receive invoice	04/01/2018 14:12:35
	Check received invoice	04/01/2018 14:12:42
	Request data	07/01/2018 12:09:16
	Check contract conditions	07/01/2018 12:10:25
	...	...

Table 2.1: An example event log with Case ID, Activity and Timestamp

To give an example, Table 2.1 shows two different executions of *Check received invoice*. The throughput time of *Check received invoice* is calculates as the time between its timestamp and the timestamp of the next activity. In Case 1, *Check received invoice* takes 51 hours and 10 seconds to complete and is proceeded by *Final check of invoice*. In Case 2, *Check received invoice* takes 69 hours, 56 minutes, and 34 seconds to complete and is proceeded by *Request data*. We can determine that these 2 executions of *Check received invoice* clearly deviate from each other. This small snippet of events therefore raise many important business process questions:

- Does the throughput time of *Check received invoice* comply with our *Service Level Agreements* (SLA) or *Key Performance Indicators* (KPIs) in both *cases*?
- Why is *Check received invoice* in Case 2 slower than Case 1?
- Does the order of performed activities affect the throughput time of *Check received invoice*?

Event logs can contain much more information related to the business processes. For example, which employee worked on the activity, which machine was involved, which department performed the activity, what type of activity was performed, which client was this activity for, etc. This information can be used to discover possible *root causes* of bottlenecks (e.g., particular invoices are more complicated and require more time).

2.1.3 Minimum viable event log

The desired input for this thesis is an event log, we consider the *minimal viable event log* to be the same event log structure as Table 2.1. That is, an event log that holds atleast the following three attributes: *Case ID, Activity* and *Timestamp*. Later in this thesis, we define several techniques for enhancing the two most common type of nonstandard event logs which deviate from this format, such that they can also be used in our research.

2.2 Time Perspective in process Mining

For the purposes of this research, focus will be on the time perspective of process mining, as discussed in Section 1.4. In this section, we explain the current execution and usage of time perspective in process mining. We do this to show what insights can be gained from the time perspective of process mining. According to chapter 9.4 of *Process Mining* [5], the time perspective of process mining provides to following performance related information:

Type 1: Analysis of frequencies and utilization, using model and frequency information to determine routing probabilities.

Type 2: Visualization of waiting and service times, using statistics such as average waiting time for an activity

Type 3: Flow time and SLA analysis, using measurement from activity x to activity y to check adherence to agreements made

Type 4: Bottleneck detection and analysis, using highlights to annotate where the most time is spend in the system, and analyzing separately to find root causes for the delays.

The following example shows a simple business process that involves the payment of incoming invoices. As can be seen in Figure 2.1 and 2.2, this process follows 2 distinct flows, namely:

Flow 1: Receive invoice → Check received invoice → Final check of invoice → Approve Invoice → Pay invoice

Flow 2: Receive invoice → Check received invoice → *Request data* → *Check contract conditions* → Final check of invoice → Approve Invoice → Pay invoice

The information presented in Figure 2.1 and 2.2 is used by business analysts to extract performance related information in the following way:

Type 1: Flow 1 is utilized 871 times, while Flow 2 is utilized 383 times. Flow 1 is utilized 69% of the time, while Flow 2 is utilized 31% of the time (Figure 2.1)

Type 2: Flow 1 takes ∼4.7 days on average (3m + 1.7d + 1.8d + 1.2d), while Flow 2 takes ∼8.7 days on average (3m + 2.7d + 16s + 3d + 1.8d + 1.2d) (Figure 2.2)

Type 3: Using the information from 1 and 2, we can check whether our business process is adhering to flow times and SLA's we defined with our clients.

Type 4: Using the information from 1, 2 and 3, we can check whether our business process is performing as we expect. If not, what root cause can we find for this?

Performance (information) consists of several different types. These types all utilize different parts of the information present in an event log. In this thesis, when we mention performance (information), we mean Type 2 performance (information). This definition is very similar to the one made by Leemans et al. [12], where performance (information) is defined as service time (the time a resource is busy with a task), waiting time (the time between an activity becoming enabled and a resource starting to execute it), sojourn time (the sum of both) and synchronisation time (for concurrent activities, the time between completion of the first and completion of the last).

From the relatively simple view presented in Figure 2.1 and 2.2, quite some information on this business process can be gathered. This could explain why the discipline has been gaining much attention from the governmental and corporate sectors [5, 13, 14]. In Chapter 3, we will discuss the limitations of this type of performance analysis.

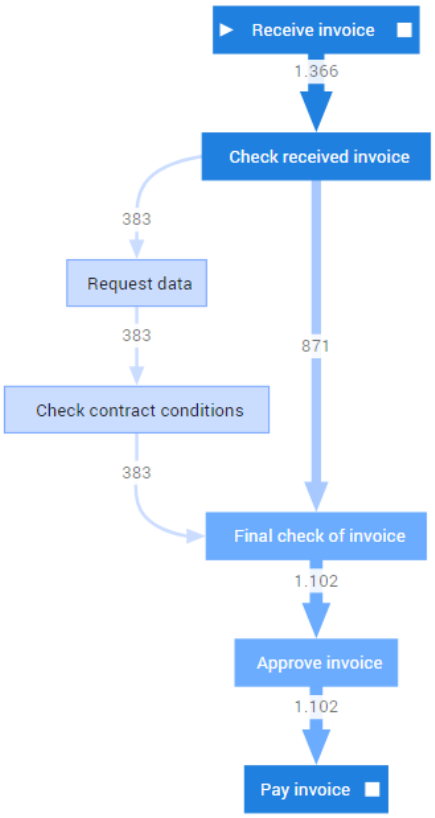

Figure 2.1: Visualized event log using a *directly-follows graph*, edges are annotated with the number of *cases* that pass through it

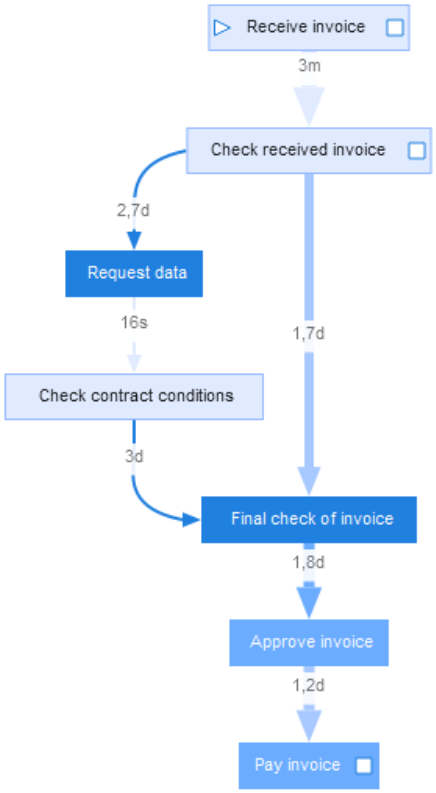

Figure 2.2: Visualized event log using a *directly-follows graph*, edges are annotated with the average throughput time per activity

2.3 Segments in event logs

This thesis analyzes the performance of *segments* in event logs. In Section 2.3.1, we explain how a segment is defined. This definition is based on research done by Fahland et al. [11] (which is discussed in Section 2.3.3). In Section 2.3.2, we explain how *segment* are detected within event logs, and how performance information is extracted from these *segments*.

2.3.1 What is a segment?

The definition of a *segment* is based on a data structure defined in research by Fahland et al. [11]. To provide an example of how a *segment* is defined, we first visualize the event log as a flow model. The flow model can be used to show the control-flow of a (sub)set of *cases*. Shown in Figure 2.3 is the flow model of Case 1 from the event log of Table 2.1.

We define a *Segment* as the flow from one activity, to an activity that directly follows it. From table 2.1, the trace of Case 1 is <*Receive invoice*, *Check received invoice*, *Final check of invoice*, *Approve invoice*, *Pay invoice*>, which in turn translates to the segments (*Receive invoice* → *Check received invoice*), (*Check received invoice* → *Final check of invoice*), (*Final check of invoice* → *Approve invoice*), (*Approve invoice* → *Pay invoice*). The *time interval* of a segment is calculated as the difference between the timestamps of the two events. So for segment *Check received invoice* → *Final check of invoice*, the *time interval* is 2 days, 3 hours and 10 seconds. For *Final check of invoice* → *Approve Invoice*, the *time interval* is 1 day, 20 hours, 35 minutes and 3 seconds, etc. The *time interval* can be of different types, namely, in seconds, minutes, hours, or days. We chose to convert every *time interval* type to seconds, since this can be easily done. Furthermore, seconds are the smallest practical unit of time to perform analysis with. We chose seconds because counting in milliseconds produces impractical large numbers when dealing with activities that take longer than a few hours.

2.3.2 Detecting segments and extracting performance information

For the event log specified in Table 2.1, a list of every segment within it can easily be determined. Event logs consist of *cases* which hold multiple (successive) events. To extract which segments are in an event log, we traverse every case in the event log and record every pair of events < A, B > where B directly follows A. Furthermore, the *time interval* is also recorded. We show an example segment table in Table 2.2.

Any case can be part of one or more segments. For example, Case 1 appears in four different

Segment	Properties	
	Case ID	Time interval (seconds)
Receive invoice → Check received invoice	Case 1	2
	Case 2	25
	Case 14	18
	...	...
Check received invoice → Final check of invoice	Case 1	183610
	Case 2	146120
	...	...
Final check of invoice → Approve invoice	Case 1	160503
Approve invoice → Pay Invoice	Case 1	111039
	Case 2	122320
	...	...

Table 2.2: Segment table after detection

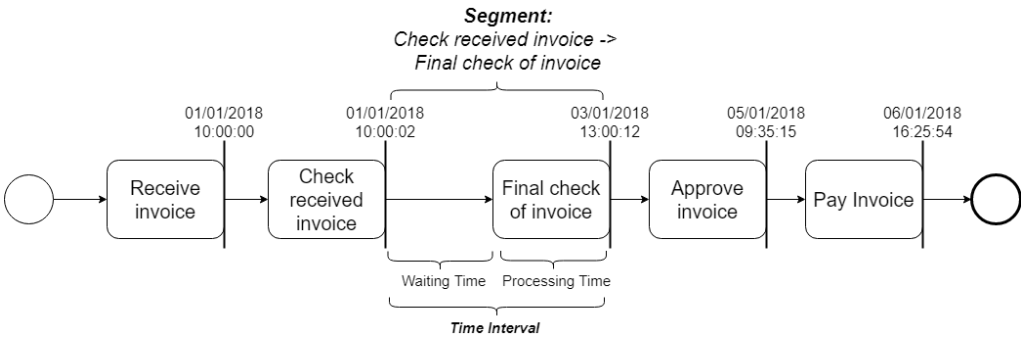

Figure 2.3: *Segment* and *time interval*, when event logs only contain end times

segments in Table 2.2. Alternatively, any segment can contain one or more *cases*. The event log shown in Table 2.1 contains only end times. Therefore, we cannot know exactly how the difference in time between two activities is distributed. As can seen in Figure 2.3, the *time interval* is calculated as the throughput time of the activity after the arrow in the pair (so for segment (A $\rightarrow$ B), the time interval is calculated as the throughput time of activity B).

2.3.3 Performance Spectrum Miner

The segment extraction technique is based on research done by Fahland et al. [11]. This research explores process performance measures that do not rely on *prior aggregation*, and provides a novel visualization of process performance data. In Figure 2.4, we see an example of the Performance Spectrum Miner (PSM) application being applied on an event log. On the left side of Figure 2.4, we see an example of a widely used technique for visual performance analysis of event logs, namely, using a graph-based model (as discussed in Section 2.2). This graph-based model shows no temporal patterns or changes over time, just aggregated (i.e., min, max, median, etc.) performance information. The right side of Figure 2.4 displays the performance spectrum of the data used to discover the model on the left side. The performance processes event log data into *segments* (as discussed in Section 2.3.2), whereafter the performance of each occurrence of the segment is measured, and plotted onto a timeline. The resulting performance spectra reveal several non-stationary of performance and synchronization of different *cases* over time. Additionally, the visualization shows that different *cases* perform differently due to systematic and unsystematic *variability in performance*. The research highlights the presence of systematic *variability in performance* in event logs. The exploration and utilization of this phenomenon is the main topic of this thesis.

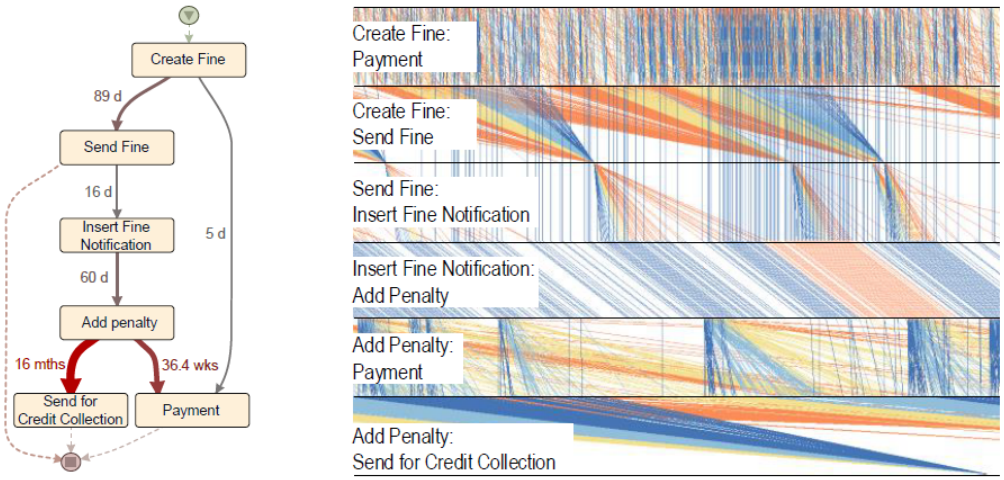

Figure 2.4: Performance analysis, using Graph based model (left) and Performance Spectrum Miner (right)

2.4 Frequency analysis of case attributes

One of the perspective of process mining discussed in Section 1.4, is the *case perspective* [6]. The *case perspective* focuses on the properties (data attributes) of a case. Such properties could be: supplier, order type, loan reason, risk factor, etc. Recent research done by Verhoef [15], provides a new way of analysing case attributes. This research was done as part of an internship assignment at ProcessGold. The goal of this internship was to integrate case attribute *frequency analysis* into the platform. Case attribute *frequency analysis* is a type of root-cause analysis which analyzes whether the observed number of case attribute values, is significantly different than the expected number of case attribute values. The implementation was based on the presence of *tags* within *cases*. A *tag* is used to identify whether a case does, or does not, comply with certain rules set up by the company. For example, a tag could denote whether a manufactured product was constructed within tolerances. To explain how the *frequency analysis* works, we present the following example. We have an even log which consist of *cases*. These *cases* have an attribute named 'Manufacturer', which denotes who manufactured the product. The attribute can have the value 'Manufacturer A' or 'Manufacturer B'. Additionally, the *cases* have a *tag* called 'Within tolerance', which denotes whether the product was manufactured within tolerances. The *tag* can have the value 'True' or 'False'. We observe that 90% of the *cases* have the *tag* 'Within tolerance' set to 'True', the other 10% is set to 'False'. Now, if we take all *cases* which were handled by 'Manufacturer A', we expect to see that 90% of *cases* have the tag 'Within tolerance' set to 'True', and 10% set to 'False'. We now use *frequency analysis* to check whether the observed frequency of this tag is significantly different from the expected frequency of this tag. So, for example, we could observe that *cases* of 'Manufacturer A' are set as 60% 'True' and 40% 'False'. *Frequency analysis* shows that this is significantly more than the expected average. This discrepancy could signify that 'Manufacturer A' is under performing, and is a bottleneck. This type of frequency analysis fits really well with our research. Therefore, this research is implemented in the visualization of our research (albeit slightly adapted to work with our data type). This is discussed in Section 7.3.

2.5 Process mining perspectives in the ProcessGold Application

The ProcessGold application contains many template dashboards, each which visualize a part of the process mining perspectives discussed in Section 1.4. Further in this thesis, we base two of

our visualizations on these template dashboards. The first dashboard we base our visualization on is the *Process analysis* dashboard of the ProcessGold application. This dashboard visualizes (amongst other things) the *organizational perspective* of process mining. In Figure 2.5, the user can select which event attribute they want to analyze. Currently, the application is limited to showing the event attribute frequency with respect to their appearance in activities (i.e., for every activity, the application calculates how frequently the event attribute appears in that activity).

The second dashboard we base our visualization on is the *Process Compare* dashboard of the ProcessGold application. This dashboard visualizes (amongst other things) the *control-flow* of process mining. The dashboard provides process-flow analysis capabilities in the form of a process graph viewer. This viewer has the functionality to visually compare two processes flows to each other, and display the similarities and differences between them. We show the process graph viewer in Figure 2.6.

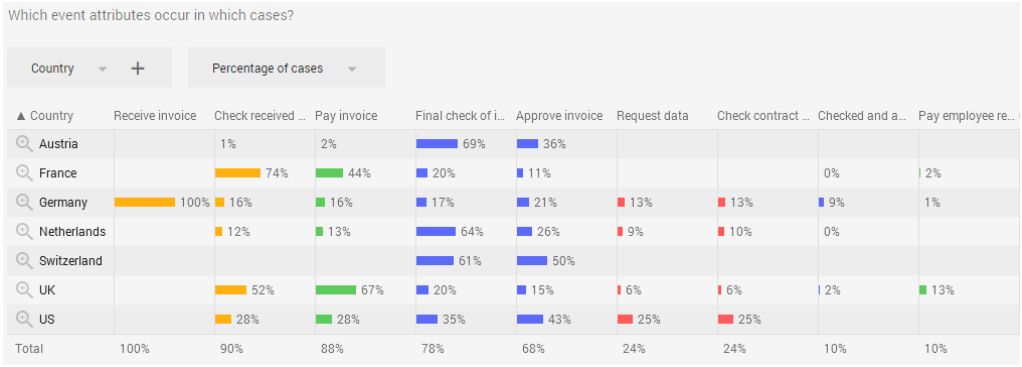

Figure 2.5: Event attribute viewer in the ProcessGold Application

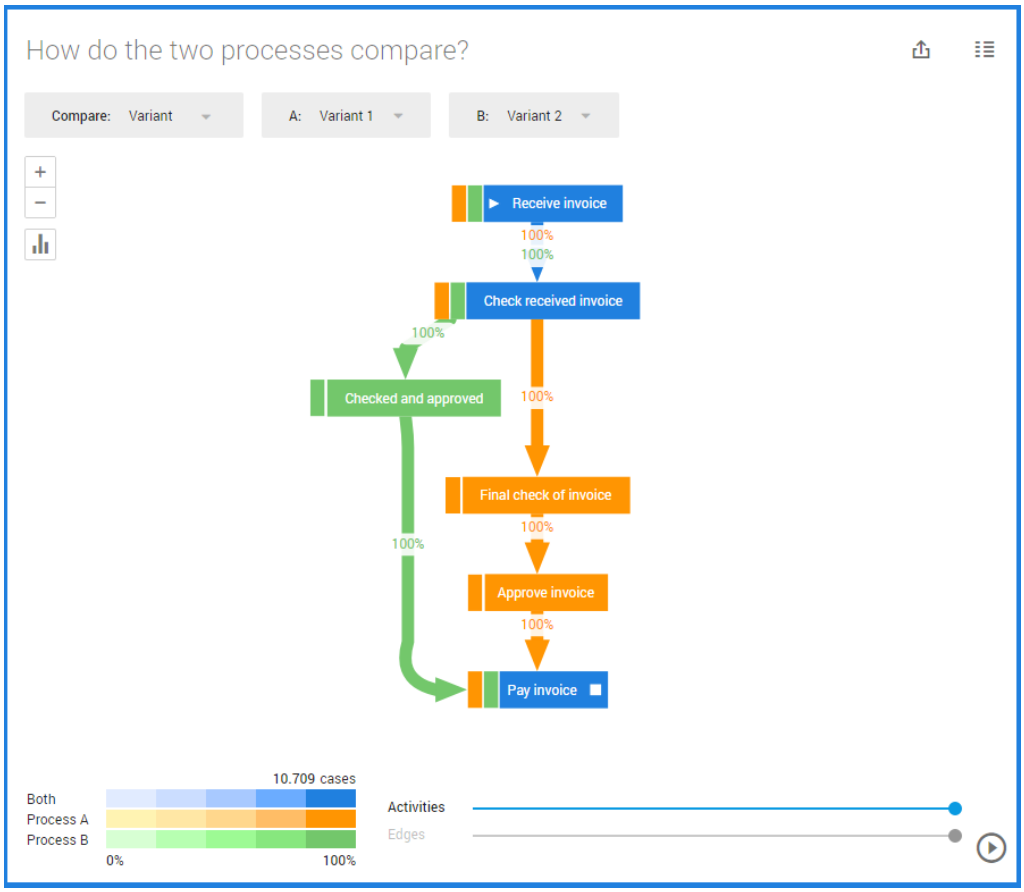

Figure 2.6: Graph viewer in the ProcessGold Application

Chapter 3

Research Motivation and Goals

The purpose of this chapter is to set up, explore, and identify a precise problem definition, together with our research motivation and goals. First, in Section 3.1, we explore the limitations of the current approach to the *timing analysis* presented in Section 2.2, and explain how this approach can sometimes misinterpret performance information. In Section 3.2, we explore and explain a technique described by Fahland et al. [11] which highlights the limitation and misinterpretation of current timing analysis approaches. The paper proposes a novel visualization of process performance data in event logs, in the form of the Performance Spectrum Miner (PSM). Furthermore, we highlight the shortcomings of this approach, by addressing several limitations. Finally, in Section 3.3, we combine the knowledge gap highlighted by the PSM, together with the limitations of the PSM, into a main research question. We present how this research question can be divided into smaller secondary research questions, and what our approach is to solve these research questions.

3.1 Limitations of current timing analysis

Current descriptive performance analysis is quite mature within process mining. Usually, the analysis is done by aggregating measures over a certain period of time, using for example, maximum or average waiting times. This descriptive performance analysis however, does not account for fluctuations in performance. Taking maximums and averages of a set measurements will obfuscate any information about possible interactions, distributions and details of these measurements.

Process mining [5] explains a common way of presenting performance related information of *timing perspective* type 2 and 4. Usually, it is done by annotating the average throughput times on their respective edges. Furthermore, highlighting the activities where the most time is spend, by giving that activity a different color. An example of this can be found in Figure 3.1.

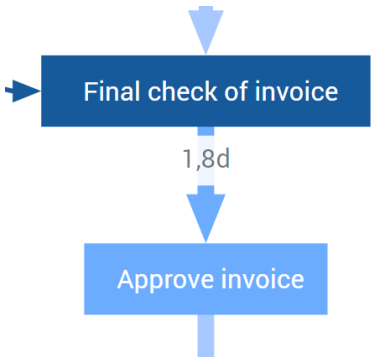

Figure 3.1: Subsection of the directly-follows graph of Figure 2.2

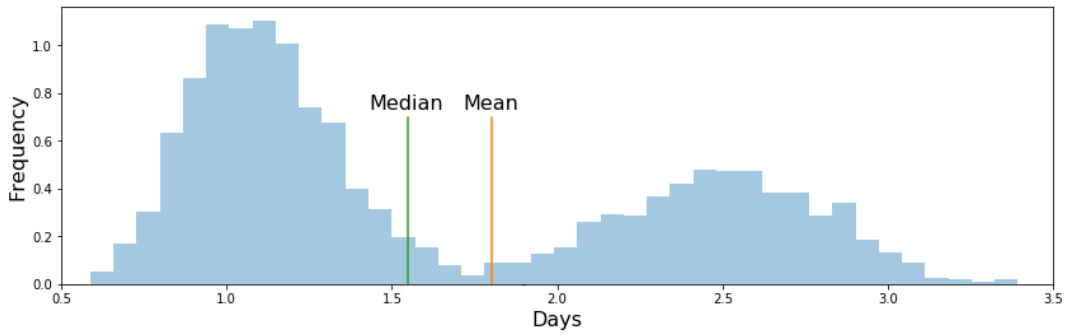

Figure 3.2: Distribution of segment A

In this thesis, we see a *segment* as a pair of activities (A,B), where activity B directly follows activity A. In this example, we are interested in the segment *Final check of invoice* → *Approve invoice* (hereafter referred to as *segment A*). Denoted on the the edge is the average throughput time (average of all cases that pass through that edge). In this example, the average throughput time is 1.8 days.

However, when we look at the actual distribution of throughput times of segment A in Figure 3.2, we see some interesting differences. While the graph from Figure 3.1 only shows aggregated information (average: 1.8 days, median: 1.55 days) in segment A, the actual distribution of this segment reveals substantially more information. Not only does it show that the data is spread quite for apart from the average and median, it also shows 2 distinct performance classes (peaks) of cases going though this segment. The left peak resembles a log-normal distribution, while the right peak resembles a normal distribution. This means that segment A has a *multi-modal* distribution. Further analysis on additional datasets reveal that many segments contain two or more distinct performance classes, and therefore consist of many multimodal distributions. This shows that currently a substantial amount of (valuable) information is lost by applying currently standardized process mining techniques.

In this section, we showed that *variability in performance* is present within event logs. In the next section, we explore research, which shows that this *variability in performance* is not a single occurrence, but systematic within event logs.

3.2 Performance Spectrum Miner

In the paper "Unbiased, Fine-Grained Description of Processes Performance from Event Data" [11] research is done on process performance measures *without prior aggregation*. The paper sets out to "provide a comprehensive of raw process behavior without enforcing prior aggregation of data". The paper starts by structuring process data into segments, and creates a performance spectrum of each segment. To provide an example, we run the event log of Table 2.1 through the Performance Spectrum Miner (PSM). The application creates a *performance spectrum* for each segment in the event log. In Figure 3.3, we show the performance spectrum of the segment *Final check invoice* → *Approve invoice*. In the *performance spectrum*, the x-axis is a timeline. Each case that passes through this particular segment is drawn as a line. For every case, the PSM takes the timestamp of *Final check invoice*, and the timestamp of *Approve invoice*, and draws a line between these two points. The colour of a line denotes to how the case performance (i.e. how fast the case goes though the segment) relative to other cases. There are four colours: Blue, light blue, yellow, and orange. Each colour holds 25% cases of the dataset. In the example, the colour blue means the case is in the top 25% fastest cases, light blue means the case belongs to the next 25% cases (25% - 50%), yellow means the case belongs to the next 25% cases (50% - 75%), and orange

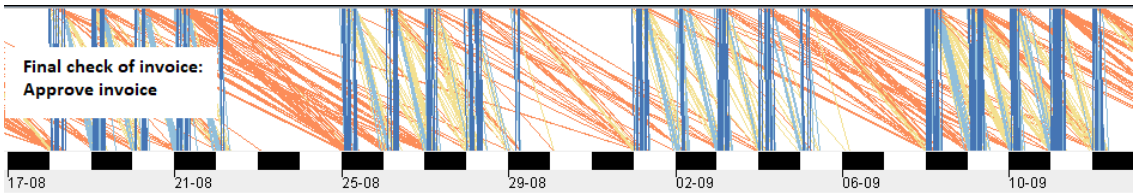

Figure 3.3: Performance spectrum of segment *Final check invoice* → *Approve invoice*

means the case belongs to the final 25% cases (75% - 100%).
In Figure 3.3, we see that the segment *Final check invoice* → *Approve invoice* has significant variability in performance. First, we see a five day workweek, indicated by the five consecutive blue pillars. Additionally, some of the cases go through the segment in a few hours, others take several days to complete it. Additionally, we see that many cases are passed towards the next workweek. All of this information is not visible in the weighted DFG of Figure 3.1, since it only shows that the segment took 1.8 days *on average* to complete the segment. This means it that the weighted DFG provides very limited insight into the actual performance of the segments. Alternatively, the PSM shows that the the actual performance of this segment varies significantly.

Current process mining analysis techniques can lead to wrong conclusions. To provide an example, assume the segment of *Final check invoice* → *Approve invoice* has an SLA which says it has to be completed within 2 days. Using the DFG on Figure 3.1, an analyst would assume the cases are inline with this SLA, since the DFG shows that the segment takes only 1.8 days to complete. However, the PSM of Figure 3.3 shows numerous breaches of this SLA, where many cases take more than 2 days to complete the segment. Therefore, Figure 3.3 provides an analyst with better insight into the actual performance distribution of this segment. Additionally, not only does the performance spectrum show segment performance, it also shows segment characteristics through the patterns it generates. These patterns can be FIFO, LIFO, Batching, Variable speed, Constant speed, etc. These characteristics are not shown in the DFG of Figure 3.1.

Fahland et al. [11] analyzed 12 event logs using the PSM. They concluded that "Process performance is neither stationary nor are cases isolated from each other". Additionally, they highlighted that current process mining techniques give only limited insight into performance data. They showed that *variability in performance* does not occur randomly throughout an event log, but appears to be a systematic phenomenon in event logs.

3.2.1 Limitations of the PSM

While the PSM gives an interesting new perspective in process performance analysis, there are several limitations to visualization through the PSM:

1. The assessment of whether a segment *has significant variability* has to be made **by a human**.
2. The PSM contains an overwhelming amount of information
3. There is no easy way to compare cases with different performance characteristics

We start with limitation 1. Some performance spectra consist of only the same throughput time, for example, when a segment is automated, such that every iteration takes an exact amount of time. This highlights the limitation that the assessment of whether a segment *has significant variability* has to be made **by a human**. The analysis therefore has to be done manually, which is not preferable when knowing that event logs contain hundreds of segments and thousands of cases.

For limitation 2, it is clear that PSM contains an overwhelming amount of information. The current presentation of this information is neither easily readable nor understandable for the untrained eye. For any business analyst, it is very important that such information is presented in a clear and interpretable way.

Finally, for limitation 3, there is no easy way to compare cases with different performance characteristics (which go through the same segment) to each other. Such a comparison is essential when business analysts want to perform bottleneck identification, since that information is essential for highlighting which characteristics cause the difference/instability.

3.3 Problem definition and research questions

In Section 3.1, we showed that *variability in performance* exists within event logs. In Section 3.2, we showed that *variability in performance* is a systematic phenomenon in event logs. Additionally, we showed that segments consist of several interesting performance characteristics when using non-aggregated data of over time. Finally, we described several additional limitations with the PSM in the previous section, which currently inhibit proper bottleneck identification.

The (automatic) extraction of performance information and analysis of the *variability in performance* has not been researched thus far. This is a gap within process mining knowledge, and discovery of this knowledge could possibly be used to facilitate bottleneck identification. To try and discover this knowledge, we set up to following main research question.

RQ: Can variability in performance be automatically discovered, analyzed, and visualized to facilitate bottleneck identification?

To answer the main research question, several secondary research questions have to be answered. The first of these questions is a follows:

RQ 1: Can performance information, extracted from event log segments, be (automatically) subdivided into different performance classes which each represent a part of the variability in performance?

As discussed in Section 3.2, *variability in performance* is present in event data. This variability shows itself on a segment level. In order to answer **RQ 1**, we have to answer the following subproblems. First, we need to discover every segment in an event log. For every segment found, we extract and store which cases go through the segment, and what performance information these cases have in that segment. Secondly, we need to subdivide them into several performance classes, which each represent a part of variability in performance. To classify the performance information, several different clustering/segmentation techniques for numeric data have to be explored and compared. The technique should be able to automatically subdivide performance information into classes, such that it subdivides the *variability in performance*. Once a satisfactory technique has been found, it is adapted to properly work with event data.

For each the classified performance classes, it is important to discover what makes them similar/different from other performance classes (within the same segment). We can gain valuable insights by comparing performance classes and determining why some classes are performing worse/better than others. Furthermore, it helps us to understand the relation of one class to the process overall. For these reasons, we create the second secondary research question:

RQ 2: Can a set of analyses be created, as to discover similarities/differences within and between the performance classes?

To answer **RQ 2**, we need to discover, test and compare several statistical analysis techniques. These techniques should be able to distinguish several different performance characteristics, such

as size, speed, frequency, distribution etc.

To ultimately support bottleneck identification, all information found in the previous step needs to be presented as to be understandable and usable for business analysts. For this, we create the third and final secondary research question:

RQ 3: Can the outcome of the performance class analysis be aggregated and visualized in such a way, as to support bottleneck identification?

To answer **RQ 3**, we need to discover which (sub)set of techniques from the previous step have the highest probability of signifying bottlenecks. There are two steps taken to validate whether these techniques actually signifying bottlenecks. First, the techniques are applied to several BPI Challenges [16], whereafter the findings are compared to the reports of the BPI Challenge winners (academic and professional categories). If the techniques reach conclusions similar to the conclusions in the BPI reports, we confirm that the techniques are able to identify bottlenecks. Furthermore, the outcome of the techniques need to be presented into an interpretable way, such that a business analyst can understand, and gain (bottleneck) knowledge from it. Several validation tests should be constructed and performed, such that we can validate whether these visualizations support analysts in discovering bottlenecks. This is done by cross-referencing whether the analysts are able to pinpoint bottlenecks with the support of the visualization better, than without the support of the visualization.

Chapter 4

Approach

This chapter aims to present an overview of the steps needed to answer the **RQ:** *Can variability in performance be automatically discovered, analyzed, and visualized to facilitate bottleneck identification?* We do by answering the secondary RQs defined in Chapter 3.3. We answer these questions by first defining the (sub)problems that need to be solved. Secondly, we present one or more methods which aim to solve these problems. Finally, we present which solutions are discovered and what follow-up steps are taken. In Section 4.1, we provide the approach for answering **RQ 1**. In Section 4.2, we provide the approach for answering **RQ 2**. Finally, in Section 4.3, we provide the approach for answering **RQ 3**.

4.1 Extraction and automatic classification of performance information

This section aims to provide an overview for answering **RQ 1:** *Can performance information, extracted from event log segments, be (automatically) subdivided into different performance classes which each represent a part of the variability in performance?*

We summarize the method for answering **RQ 1**, the details of which are given in Chapter 5. The research question consist of 2 parts. The first part is the extraction of performance information from event logs, per segment. We defined which information we need from event logs based on the motivation of Section 3.2. This information consists of an event log with a Case ID, Activity and Timestamp (i.e. minimum viable log), explained in Section 2.1.3. In Section 5.1, we present several techniques for converting the two most common types of event logs which do not adhere to our required format. After obtaining our required event log structure, we detect which segments are present within an event log, which we explain in Section 5.2.

The second part of the research question requires us to identify and adapt a method to (automatically) subdivide performance information, into several difference classes, such that each part represents a part of the variability in performance. The entire structure is discussed in Section 5.3. We start by first creating several criteria/requirements to which the method needs to adhere in Section 5.3.1. These requirements are based on the characteristics of our input data, and the desired form in which these need to go. These criteria are:

(1) The method should be able to handle positive number
(2) The method should not make assumptions on what the distribution of its input data is
(3) The method should not determine/restrict the size of the class, just determine whether two datapoints belong to the same class.

In Section 5.3, we look towards literature to discover several existing techniques for separating

performance information into distinct classes. The techniques we identified are: Equal Interval, Quantile [17], Head/Tail Breaks [18], k-means clustering [19] and Jenks Natural Breaks Optimisation [20]. Both Equal Interval and Jenks Natural Breaks Optimisation satisfied the criteria. Therefore, we test both techniques on our data and inspect the results manually. We observe that using Equal Intervals can result in unwanted behaviour (i.e. splitting a peak down the middle). Furthermore, Osaragi [21] argues that a classification method which keeps the loss of inherent information as low as possible can be considered an effective method, since it lessens the error of judgment. Osaragi also showed that Equal Interval classification comes with significantly more information loss compared to Jenks Natural Breaks Optimisation.

From the discoveries in Section 5.3, we decide that *Jenks Natural breaks* (hereafter referred to as *Jenks*) [20] is the classification method that suits our purposes best. *Jenks* tries to divide data into several classes, such that the variance within each class is minimized, and the variance between each class is maximized. An overview of how *Jenks* works is given Section 5.4. Additionally, to create a clear picture of how *Jenks* is applied in the context of performance information, we give an in-depth example of applying *Jenks* on performance data in Section 5.4.1.

Our goal for this sub RQ is to *automatically* subdivide performance information into different performance classes. For this, we need to approximate what an optimal number of subdivisions is for a particular dataset. However, *Jenks* requires the number of classes in which the data is to be classified, as an input parameter. In Section 5.4.2, we explain how output from *Jenks* is used to *automatically* approximate an optimal number of classes. For this method to work, it requires one input parameter to be set beforehand. How the value of this parameter is set, alongside an argument why, is also discussed in Section 5.4.2.

Finally, in Section 5.5, we try to create an understanding of the output that is created by the techniques explained in Section 5.1 through 5.4. We do this by providing an example using real data from BPI2017 [22].

4.2 Analysis, extension and classification of performance classes

This section aims to provide an overview for answering **RQ 2:** *Can a set of analyses be created, as to discover similarities/differences within and between the performance classes?*

In Section 4.1, we discussed how performance information is extracted from an event log, and how to classify this information into distinct performance classes. We also discussed how performance information is extracted from an event log *per segment*. In this section, we summarize the method for answering **RQ 2**, the details of which are given in Chapter 6. In Section 6.1, we explain how a *segment* is part of a larger *overall process* (determined by the cases that make up a segment). The *segment performance classes* consist of a set of cases, and their respective *segment time*. To generate *overall performance classes*, we determine, for each case in a class, what the total throughput time of that case is (i.e., time between the start and end of the case). In the *overall performance classes*, each case still belongs to the some class, however, we replace the *segment time* of the case with *total time* of the case. This generates a new *overall performance class* distribution shown in Figure 6.3.

Now that we have our *segment* and *overall* performance classes. We would like to understand how these performance classes relate to each other. In a perfect world, every case should be able to pass a *segment* equally fast. However, various reasons (i.e., a difficult cases, different event/case attributes, external factors) cause some cases to perform worse in the same *segment*. To properly assist analysts, we want to *automatically* analyze each *segment*, determine which *segments* are

most likely bottlenecks (i.e. segments which negatively influence the throughput time of a process), and therefore to focus their improvement efforts on. To address this, we need to discover a set of *measures*, which quantify the characteristics of performance classes. For simplicity sake, we define performance class characteristics as *measures* for the rest of this thesis. We have two types of measures: Measures **within** *segment/overall* performance classes, and measures **between** *segment/overall* performance classes.

Measures **within** *segment/overall* performance classes:

1. Do the performance classes have significant overlap, or not? (Effect size)
2. What is the time difference between the slowest and fastest classes? (Potential lost time)

Measures **between** *segment/overall* performance classes:

3. What is the influence of the segment time, with respect to the total time? (Total weighted impact)

We want to be able to automatically collect the measures from the segment classes. Therefore, in Section 6.2, we propose several algorithmic solutions which quantify these measures, while keeping in mind that we have a specific data type, namely segment data. To determine whether the performance classes has significant overlap (measure 1), we have to find a measure which calculates class overlap (i.e., effect size). We determine that *Cohens d* [23] is a suitable statistical analysis technique to measure class overlap. Because *Cohens d* is a measure between two classes, and our classification can produce more than two classes, we create a custom algorithmic solution for determining the *average effect size* of *segment/overall* performance classes. For determining what the time difference between the slowest and fastest classes is (measure 2), there is no standardized statistical measure. Therefore, we develop a custom algorithmic solution to determine this measure. For determining what the influence of segment time has on the total time (measure 3), there is also no standardized statistical measure. Therefore, we also develop a custom algorithmic solution to determine this measure.

Now that we have the ability to calculate all three measures for each segment. We should create a way to rank them such that we can determine which segments are most likely bottlenecks (or atleast have interesting characteristics). Each measure can be used to form an opinion on whether a segment is more interesting to analyze *relative to other segments* (e.g., a segment can have the highest *effect size* compared to the other segments, making that segment relatively more interesting). Therefore, in Section 6.3, we first rank the segments for each measure separately. So, for each measure, every segment gets assigned a number, based on the magnitude of that measure. Every segment gets a number from one to the number of segments analyzed. The higher the number, the higher that segment ranks with respect to other segments, as shown in Figure 6.4. Now, for each segment, we sum the rankings they get for each measure, to create a *cumulative ranking*, as shown in Figure 6.4. The *cumulative ranking* is subsequently used as input for the visualization of Chapter 7.

4.3 Assisted bottleneck identification

This section aims to provide an overview for answering **RQ 3:** *Can the outcome of the performance class analysis be aggregated and visualized in such a way, as to support bottleneck identification?*

In Section 1.2, we discussed four different perspectives that are present in process mining. To recap, these perspectives are *time*, *case*, *organization*, and *control-flow*. Each perspective highlights a different part of the wealth of information event logs hold. To assist a business analyst with bottleneck identification, we want them to be able to inspect their data from each of these perspectives. In this section, we summarize the method for answering **RQ 3**, the details of which

are given in Chapter 7 and 8.

In Chapter 7, our goal is to build a set of dashboards where each board highlights one of the four perspectives of process mining. This is to assist the analyst with bottleneck identification. To do this, we first explored the default dashboards that the ProcessGold application has, and noted which useful visualizations it already provides. With feedback from ProcessGold Solution Engineers, we either extended the default dashboards, or custom developed new dashboards, each of which highlight a different process mining perspectives. In Section 7.1 and 7.2, we implemented two dashboards which visualize the *time perspective*, by providing an overview of segments, and tools for deeper analysis. In Section 7.3, we visualize the *case perspective*. We do this by using the *frequency analysis* (as discussed in section 2.4) functionally that was already implemented in the ProcessGold platform by Verhoef [15]. We slightly adapt it to work with our segment data. In Section 7.4, we visualize the *organizational perspective*. We do this by adapting the template Event attribute dashboard discussed in Section 2.5. This dashboard is severely limited by the fact that is can only show event attributes with respect to activities. Therefore, we extend it such that it allows for the comparison of event attribute frequency between classes. Finally, in Section 7.5 we visualize the *control-flow perspective*, by extending the process flow analysis dashboard (discussed in Section 2.5) to be able to work with segment data.

A business analyst is now able to view our segment analysis from every perspective within process mining. While this does not provide any guarantees that root causes for bottlenecks will be discovered, we do provide the analyst with an extensive set of tools to facilitate bottleneck identification.

Additionally, In Chapter 8, we demonstrate the functionality and correctness of the application, by applying our *segment analysis* on a Business Process Intelligence Challenge (BPIC) dataset, and comparing our findings with those of the BPIC reports. The BPIC dataset that we demonstrate the functionality on is the BPIC 2017 [22] dataset. Additionally, we evaluate the usefulness and understandability of our *segment analysis*, by performing several validation tests with ProcessGold Analysts. The validation tests are set up as follows. A ProcessGold analyst is presented with our approach, and with a BPIC dataset (either BPIC 2017 [22], or BPIC 2018 [24]). We ask the analysts to find as many (potential) bottlenecks as they can discover in 45 minutes. Afterwards, a second test is performed, this time with a different dataset than the first test, and without our approach (just using the default ProcessGold Application). Additionally, we interview the analysts ask them about their experience with our approach. We perform these two tests with four different ProcessGold analysts (for a total of eight tests). After all tests are done, we compare the set of (potential) bottlenecks the analysts found, with and without our approach. We tests whether it is possible for analysts to analyze an event log, quicker and more thorough, using our approach, compared to when they use contemporary process mining tools.

Chapter 5

Extraction and automatic classification of performance information

This chapter is dedicated to answering **RQ 1:** *Can performance information, extracted from event log segments, be (automatically) subdivided into different performance classes which each represent a part of the variability in performance?*

In Section 2.3.2, we discuss the desired input format of event logs, alongside an explanation on how segments (pairs of activities [A,B], where activity B directly follows activity A) are defined in context of those logs. In Section 5.1, we present several techniques for converting the two most common type of nonstandard event logs, to our desired format. In Section 5.2, we discuss how to detect segments in event logs and extract performance information from these segments. In Section 5.3, we explore and discuss several different techniques for classifying cases with similar performance characteristics. In Section 5.4, we discuss which technique are used, including an explanation and exploration of its parameters. Finally, in Section 5.5, we provide an example using real data, by running a segment of the BPI 2017 dataset [22] through the classification technique.

5.1 Converting nonstandard event logs

We have defined what our desired input format in Section 2.3.2. However, event logs also can appear in different forms. The two most common ones (which differ from our preferred type) are logs with *lifecycle information*, and logs with *start and end* timestamps. To ensure that these types of logs can also be used for our analysis, we present some preprocessing steps. These preprocessing steps are explained in the following sections.

Event logs containing both *activity start* and *activity end* timestamps While most event logs only contain activity end times, it is possible to run into event logs that contain both start and end timestamps Analysis of BPI challenge logs 2012 through 2019 [25, 26, 27, 22, 24, 28] , showed that BPI 2012 and BPI 2017 [25, 22] contained start and end times. We first illustrate what such an event log looks like in Table 5.1.

To ensure proper execution, every event in the event log is to be split up, such that the that every activity *Receive invoice* with start time *Receive invoice*$_{start\_time}$ and end time *Receive invoice*$_{end\_time}$ is split up into two separate events. These event are *Receive invoice*$_{start}$ and *Receive invoice*$_{end}$, with timestamps *Receive invoice*$_{start\_time}$ and *Receive invoice*$_{end\_time}$ respectively. An

Case ID	Properties		
	Activity	Start time	End time
Case 1	Receive invoice	01/01/2018 09:00	01/01/2018 10:00
	Check received invoice	01/01/2018 11:00	01/01/2018 12:10
	Final check of invoice	02/01/2018 08:00	02/01/2018 08:17
	Approve invoice	02/01/2018 10:00	02/01/2018 13:25

Table 5.1: Event log with start and end times *before* preprocessing

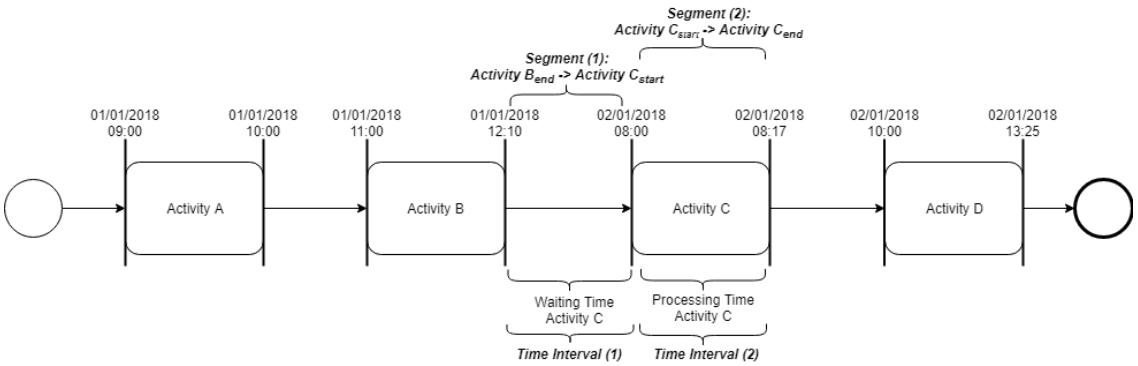

Figure 5.1: Segment *time intervals*, when event logs only contain both start_times and end_times

example is this is displayed in Table 5.2.

The event log of Table 5.2 can also be visualized using a flow diagram. Shown in Figure 5.1 is the flow diagram of Case 1 from Table 5.2. The same trace $<$*Receive invoice*, *Check received invoice*, *Final check of invoice*, *Approve invoice* $>$ has now been extended to contain the segments (*Receive invoice*$_{start}$ $\rightarrow$ *Receive invoice*$_{end}$), (*Receive invoice*$_{end}$ $\rightarrow$ *Check received invoice*$_{start}$), (*Check received invoice*$_{start}$ $\rightarrow$ *Check received invoice*$_{end}$), etc. When an event logs contains *start* and *end* times, it is possible to make a distinction between the *type* of segment that is analyzed. For example, the type of segment ($B_{start} \rightarrow B_{end}$) is 'processing time' and the *time interval* is 1 hour and 10 minutes. The type of segment ($B_{end} \rightarrow C_{start}$) is 'waiting time' and the *time interval* is 19 hours and 50 minutes.

The distinction between the *type* of segment adds to the analysis made in the later steps of this research, by enabling an analyst to distinguish between types of bottlenecks. There has been some

Case ID	Properties	
	Activity	Timestamp
Case 1	Receive invoice$_{start}$	01/01/2018 09:00
	Receive invoice$_{end}$	01/01/2018 10:00
	Check received invoice$_{start}$	01/01/2018 11:00
	Check received invoice$_{end}$	01/01/2018 12:10
	Final check of invoice$_{start}$	02/01/2018 08:00
	Final check of invoice$_{end}$	02/01/2018 08:17
	Approve invoice$_{start}$	02/01/2018 10:00
	Approve invoice$_{end}$	02/01/2018 13:25

Table 5.2: Event log with start and end times *after* preprocessing

research on *start time estimation* [29], which explore techniques to add start times to event logs which only contain end times. However, the addition of start time estimation would add (unknown) variation and uncertainty to the analysis methods used, and, therefore, would obfuscate whether the conclusions the we reach are accurate. Therefore, these estimation techniques are not included in this thesis.

Event logs with activity lifecycles Some event logs can contain attributes that denote the state of an activity, named lifecycles. These lifecycles denote what 'status' an activity is in, for example, *started*, *ended*, *suspended*, or *resumed* [5]. An example of such a log is given in Table 5.3. During this research, several event logs [25, 22] are analyzed that have such a lifecycle attribute in it. Both of these play an essential role during the implementation and validation phases, and therefore, the decision was made to explicitly state how these logs are converted.

Since our approach works using only *Case ID, Activity* and *Timestamp*, we can adapt this input into a workable form. This is done by changing the activity name into a combination of the activity and the lifecycle. An example is presented in Table 5.4.

This conversion alters the activity name, such that it contains the event log cycle within. Instead of having one activity variant, the event log now has four different variants. An example of how this changes the definition of a *segment* is presented in Figure 5.2.

The only change to the event log is that activities are renamed. This means that we can use the same method for extracting segment and calculating *time intervals* as in Section 2.3.2. However, lifecycles give us a bit more context about this process. Lifecycle information tells us when the activity is being actively worked on, and when the activity is currently suspended. Therefore, just like in the previous section, we can now distinguish between *types* of segments. For example, the segment (Receive invoice+start → Receive invoice+suspend) is categorised as 'processing time', the segment (Receive invoice+suspend → Receive invoice+resume) is categorised as 'waiting time', and finally, the segment (Receive invoice+resume → Receive invoice+complete) is again categorised as 'processing time'.

Case ID	Properties		
	Activity	Lifecycle	Timestamp
Case 1	Receive invoice	start	01/01/2018 16:00
	Receive invoice	suspend	01/01/2018 16:10
	Receive invoice	resume	01/01/2018 16:12
	Receive invoice	complete	01/01/2018 16:20

Table 5.3: Event log with lifecycles

Case ID	Properties	
	Activity	Timestamp
Case 1	Receive invoice+start	01/01/2015 16:00
	Receive invoice+suspend	01/01/2015 16:10
	Receive invoice+resume	01/01/2015 16:12
	Receive invoice+complete	01/01/2015 16:20

Table 5.4: Event log with lifecycles combined

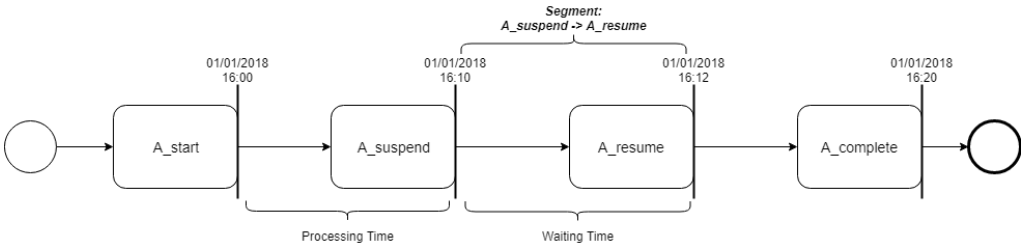

Figure 5.2: *Segment* and *time interval*, when event logs contain lifecycles

5.2 Segment detection and extraction of performance information

In Chapter 3, the *variability in performance over time* is shown to occur when analyzing the performance of a *segment*. In Section 2.3.1, we already determined what a *segment* is, and in Section 2.3.2, we explained an approach for detecting segments, and extracting *time interval* information from these segments. However, we do not consider each segment to be valid. In this section, we explain which requirements a segment needs to have to be considered valid, and why.

Later in this thesis, several different statistical analysis techniques are used on the *time intervals* of segments. These techniques require a minimum number of observations to be considered valid [30, 31, 32]. This is why we do not only record a list of every segment, but we also records how many times this segment has occurred in the event log (i.e., how many time intervals have we encountered for this segments) to be considered a valid segment. For this research, we decided the number of cases existing in a segment must therefore be atleast 50 for the segments to be considered valid. The next section explains how we use the extracted time intervals (and their respective cases) automatically classified into classes which hold similar performance information.

5.3 Automatic classification of cases with similar performance information

Recall the example of Section 3.1, where we displayed a figure of the distribution of a segment. In that section, we explained that a segment has a distribution which is not necessarily reflected by the information presented on a weighted graph, and most likely consists of a multimodal distribution. Therefore, we are looking for a way to subdivide this multimodal distribution into several classes, such that we decompose the multimodal distribution into several smaller distributions, as presented in Figure 5.3.

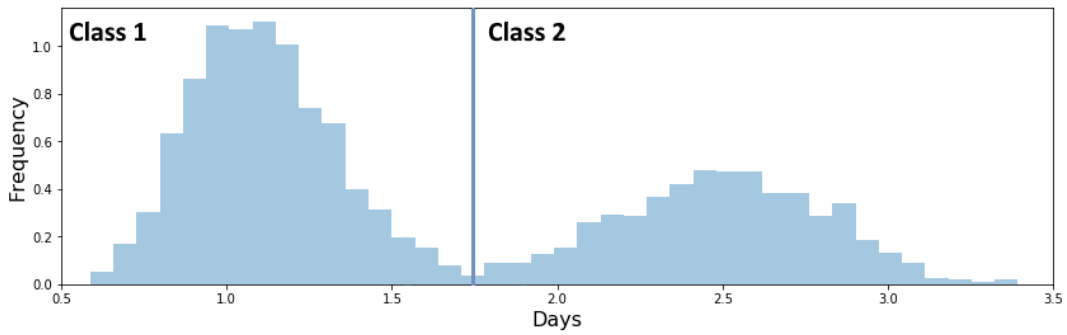

Figure 5.3: Distribution of segment Receive invoice, classed into distinct sections

5.3.1 Classification criteria and literature

To discover which methods best suit our purpose, it is important to denote what the characteristics of our input data are. First off, we are dealing with *time interval* data, in our case, this data will always be a positive number (constraint 1). Secondly, we have no knowledge of what the event log data looks like beforehand. This means we cannot make any assumptions on how the data is distributed (constraint 2). Finally, we cannot be certain how the peaks within the data will be distributed (how many elements a peak contains). This means we cannot make any restrictions on the size of any of the classes (constraint 3).

Based on these constraints, we construct the following requirements the classifications needs to adhere to:

(1) The method should be able to handle positive numbers
(2) The method should not make assumptions on what the distribution of its input data is
(3) The method should not determine/restrict the size of the class, just determine whether two datapoints belong to the same class.

In Section 5.2, we discussed a technique to extract performance information from an event log, per segment. We can use this information as input for the automated classification. Recall that the goal of the automated classification is to class cases, such that cases with similar performance characteristics are in the same classify. We looked towards the literature to discover several different classification methods for single and multi-dimensional data, including whether they satisfy these constraints. These methods have been tried and tested throughout the years. Such methods include: Equal Interval, Quantile [17], Head/Tail Breaks [18], k-means clustering [19] and Jenks Natural Breaks Optimisation [20].

In Table 5.5, we show an overview of all methods, including whether they adhere to constrains 1 to 3. We can already remove some options which violate our requirements. *Quantile* classification breaks up the data into several chucks of equal size. This method divides the data up such that each class contains the same number of elements (e.g., 4 classes that contain 25% of the data each). While this method does not take into account the distribution of the data. It may split up a distribution (peak), directly down the middle, into two separate classes. Therefore, *Quantile* is not suited for our purposes.

Head/tails breaks classification relies on the data distribution containing a heavy tail, which violates requirement 2, and is therefore not suited for our purposes. In the same vein, *K-means clustering* relies on the data being multi-dimensional, which our data is not. Therefore, *K-means clustering* is also not suited for our purposes.

This leaves us with classification methods *Equal Interval* and *Jenks Natural Breaks*. Recall that we are searching for a method which is able to split up a multimodal distribution, such that hidden information from the distributions contained within is exposed and separated. To determine which of the two methods is best suitable for this purpose, we set up several different datasets which

Classification type	**Data**	**Input assumption**	**Class size**	**Suitable**
Equal Interval	Numerical	None	Unrestricted	Yes
Quantile	Numerical	None	*Restricted*	No
Jenks Natural Breaks	Numerical	None	Unrestricted	Yes
Head/Tail Breaks	Numerical	*Tail-heavy*	Unrestricted	No
K-means Clustering	Numerical	*Multi-dimensional*	Unrestricted	No

Table 5.5: Classification types, including their constraint attributes

holds the multimodal distribution, and test which of the two methods handles these datasets better.

Equal interval or Jenks Natural Breaks
We can see how data is partitioned when using the *Equal Intervals* method in Figure 5.4. This method divides the data up into equally sized intervals (i.e. 0-200, 200-400, 400-600). *Equal Intervals* uses equal ranges. Therefore, it works best on data that is somewhat equally spread across the entire range.

Secondly, we can see how the data is distributed when using the *Jenks Natural Breaks* method in Figure 5.5. The method tries to optimize the classification by dividing data such that the variance within each class is minimized, and the variance between each class is maximized. The method takes into account that the data can have several peaks (classes) in different areas, and tries to separate them.

Since we want to capture the distributions in their entirety, *Equal Interval* does not produce our desired results. Our data consists of multimodal distributions. The classes it produces are not as good. As can be seen in Figure 5.4, this method can split a distribution (peak) directly down the middle into two separate classes. This method does neither take into account the values of the data nor the distribution of the data, and is simply a slice and dice of the data. Alternatively, *Jenks Natural Breaks* is able to capture the different distribution significantly better. As can be seen in Figure 5.5, the division created by this method manages to enclose every peak better than the other methods.

An additional criteria to consider is *information loss*. When different observations are placed within the same class, some of the information on what made these observations different from the original data is lost. Osaragi [21] argues that a classification method which keeps the loss of inherent information as low as possible can be considered an effective method, since it lessens the error of judgment. Additionally, Osaragi showed that equal interval classification comes with significantly more information loss compared to *Jenks*. This is because *equal interval* uses a human-centric approach to data classification [33] and ignores the characteristics of the data itself, resulting in significant information loss. On the contrary, *Jenks* is a far more data-centric since it uses the characteristics of the data (in-class variance) to create a classification. In this research, we want to lower *information loss* to lessen judgement errors during analysis.

Jenks Natural Breaks is better suited as method to subdivide performance information into distinct classes. In the next section, *Jenks Natural Breaks* is explained in more detail.

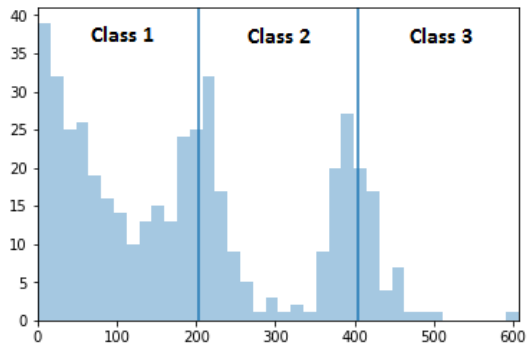

Figure 5.4: Equal interval classification

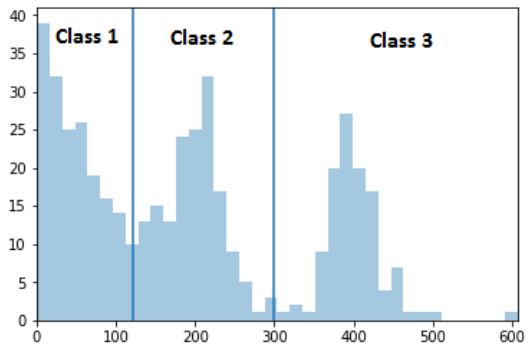

Figure 5.5: Jenks natural breaks

5.4 Jenks Natural Breaks Optimization

This section explains the execution of *Jenks Natural Breaks Optimisation* [20, 34]. *Jenks* is a one dimensional data clustering algorithm. It divides a dataset (set of numbers) into several classes by iteratively breaking up the data using different *breakpoints*.

Breakpoints are defined as follows. We want to subdivided dataset Q into n groups. For every subdivision of Q, called Q_i, a pair of breakpoints is created such that, for every breakpoint combination (bp_i, bp_{i+1}), it holds for every $x \in Q_i$, that $bp_i \leq x < bp_{i+1}$. For example, to divide $Q = [1, 2, 5, 6, 10, 12]$ into groups Q_1, Q_2, Q_3, which are $[1, 2], [5, 6], [10, 12]$ respectively, the set of breakpoints are $[1, 5, 10, 12]$.

After choosing a randomly selected set of breakpoints to start with, *Jenks* repeats the following steps, to discover the set of breakpoints with the highest *sum of squared deviations for class means* (SDCM):

Step 1: Calculate *sum of squared deviations for array mean* (SDAM) of dataset Q:
By calculating $\sum_{i=1}^{N}(Q_i - \mu)^2$, where $\mu = \overline{Q}$

Step 2: Calculate the *sum of squared deviations between classes* (SDBC) for every class Q_{bp}:
By calculating $\sum_{i=1}^{N}(Q_{bp_i} - \mu)^2$, where $\mu = \overline{Q_{bp}}$ for every $Q_{bp} \in Q$

Step 3: Calculate the *sum of squared deviations for class means* (SDCM):
By calculating SDAM - SDBC

Step 4: Repeat steps 1 - 3 for every possible breakpoint combination, and choose the combination with the highest SDCM.

We supply *Jenks* with a set of numbers Q and the desired number of classes n, it will return a set of $n + 1$ breakpoints *bp*. *Jenks* makes sure that every possible *breakpoint* combination is tested, this ensures that the division where the in-class variance is minimized, is always found. To provide better understanding of how *Jenks* works, an in-depth example is discussed in Section 5.4.1

5.4.1 Jenks Example

We use this section to provide an in-depth example of *Jenks* in the context of performance information. This is to provide a clear picture of the execution of the algorithm. In Section 5.2, we discussed how we separated cases based on which segments they hold. An example of segment extraction on an event log can be seen in Table 5.6.

Segment	Case ID	Time interval
Receive invoice → Check received invoice	Case 1	7600
	Case 2	3480
	Case 14	220
	Case 17	500
	Case 25	1500
	Case 39	1700

Table 5.6: Segments with cases their *time intervals* in seconds

Jenks starts with a list of the time intervals seen in Table 5.6, namely: $Q = [7600, 3480, 120, 600, 10000, 12000]$. The algorithm tries to create an optimal division of these number into a predetermined number of classes. For this example, we want to divide this data into three different classes. What follows are the steps *Jenks* takes to accomplish this:

Step 1: Sort the numbers and choose any arbitrary breakpoint combination of Q

Sorted Q : $[220, 500, 1500, 1700, 3480, 7600]$
breaks up into Q_{bp} : $[220], [500, 1500, 1700, 3480], [7600]$

Step 2: Calculate SDAM[1] of Q:

$\sum_{i=1}^{N}(Q_i - \mu)^2$, where $\mu = mean(Q) \implies$
$(220-2500)^2+(600-2500)^2+(1500-2500)^2+(1700-2500)^2+(3480-2500)^2+(7600-2500)^2 = 5198400 + 4000000 + 1000000 + 640000 + 960400 + 26010000 =$ **37808800**

Step 3: Calculate the SDBC[2] for every class generated from Q_{bp}

For every $Q_{bp} \in Q$, do $\sum_{i=1}^{N}(Q_{bp_i} - \mu)^2$, where $\mu = mean(Q_{bp}) \implies$

$[220] =$	$(220-220)^2$	$=$	0
$[500, 1500, 1700, 3480] =$	$(500-1795)^2 + (1500-1795)^2 + ... =$		4612300
$[7600] =$	$(7600-7600)^2$	$=$	$0 +$
			4612300

Step 4: For every class generated from the breakpoints, calculate the SDCM[3] by subtracting the SDBC from the SDAM

$SDCM$ of $[220], [500, 1500, 1700, 3480], [7600] = SDAM - SDBC = 37808800 - 4612300 =$ **33196500**

Step 5: Repeat steps 2 - 4 for every possible breakpoint combination, and choose the combination with the highest SDCM.

$SDCM$ of **[220,500,1500],[1700],[3480,7600]** $= 37808800 - 9392800 = 28416000$
$SDCM$ of **[220,500],[1500,1700],[3480,7600]** $= 37808800 - 8546400 = 29262400$
$SDCM$ of **[220],[500,1500,1700,3480],[7600]** $= 37808800 - 4612300 = 33196500$
$SDCM$ of **[220,500],[1500,1700,3480],[7600]** $= 37808800 - 2415466 = 35393334$
$SDCM$ of .....
$SDCM$ of **[220,500,1500,1700],[3480],[7600]** $= 37808800 - 1596800 =$ **36212000**

Classification with the highest SDCM results in:
Class 1: [Case 14, Case 17, Case 25, Case 39]
Class 2: [Case 2]
Class 3: [Case 1]

These steps show how to the Jenks algorithm splits up a set of numbers, into a predetermined number of 'natural' classes. We can use this functionality to split our (multimodal) *time interval* measurements into distinct classes automatically. Since these (multimodal) *time intervals* are associated with cases, we are now able to separate cases into distinct classes.

5.4.2 How to approximate the optimal number of classes?

In Section 5.4.1, we showed how *Jenks* splits up *time interval* (and therefore, cases) into several distinct classes. We know that *Jenks* requires the number of classes, to split the dataset into, as an input parameter. In this section, we explore how we can automatically approximate an optimal number of classes. We do not know how many parts (if any) the multimodal distribution of a segment consists of beforehand. This means we do not know what number of classes we should provide *Jenks* with. We need to find a way how to use *Jenks* to approximate the number sub-distributions a segment consists of. This section elaborates how Jenks can be used, such that

[1] Sum of squared deviations for array mean
[2] Sum of squared deviations between classes
[3] Sum of squared deviations for class means

an optimal number of *classes* is approximated.

Alongside *breakpoints*, *Jenks* also supplies the user with a *Goodness of Variance Fit* (GVF) measure. This measure is used to signify classification accuracy, where a GVF of 0 implies *no fit* and a GVF of 1 implies a *perfect fit*. GVF is calculated using the following calculation: (SDAM - SCDM) / SDAM. The idea behind this calculation is as follows: If the summed variance of each group (SDCM) is low, compared to the total variance of the dataset (SDAM), the GVF will be high, and the fit will be better. Simply put, the cumulative variance the groups should be smaller than variance of the original dataset. In Table 5.7, an overview is given of the GVF number, when increasing the number of classes.

Intuitively, dividing Q into three or four classes makes the most sense, since the values which lie close together, are grouped. However, this is not directly reflected by the GVF test. We can see that, due to the nature of the test, increasing the number of classes will always (to varying degrees) increase the GVF number. Therefore, simply maximizing the GVF will always create a classification where the number of classes equals the size of the input. Alternatively, we can look at the rate at which the GVF changes, In Table 5.8.

We can see that the first increase from two to three classes makes the GVF number 'improve' by 15.9%. When we increase the classes from three to four, the GVF number only improves by 0.040%, meaning that adding new classes is only significantly beneficial up to a certain point. This is why we want to calculate an additional measure, namely the *Rate of Goodness-of-Variance-Fit Change* (RGVFC). The RGVFC denotes the increase in classification accuracy, when classifying a dataset with an additional class. We calculate this measure by first supplying *Jenks* with dataset Q and a number n desired classes. *Jenks* returns a set of *breakpoints* and a GVF_n. Subsequently, we supply *Jenks* with the same dataset Q, but with $n+1$ desired classes. *Jenks* returns a set of *breakpoints* and a GVF_{n+1}. The RGVFC is calculated as $(GVF_{n+1} - GVF_n)$ / GVF_n, which is a positive value since it always holds that $GVF_{n+1} > GVF_n$. In summary, the RGVFC denotes the rate of change between GVF_n and GVF_{n+1}.

We introduce a new parameter for our algorithm, the ***RGVFC_threshold***. This parameter denotes the minimum RGVFC every additional class needs to have, before progressing. What follows is a motivation on how we determined the value for this parameter.

Determining a proper RGVFC_threshold *Jenks* is able to subdivide any set of n numbers into anywhere from two to $n-1$ classes. We have two major considerations to take into account when deciding a *RGVFC_threshold* value. First, we want *Jenks* to find and separate as many peaks as possible. We know that the multimodal distributions can come in many different forms, with most of them containing more than two distinct peaks. While we can never be fully certain we have found every distinct peak, we can give *Jenks* enough freedom to discover a significant portion of them. Secondly, we have to take into consideration that the classification made by *Jenks* will ultimately have to be shown to an analyst. Research has been done on the limit of human mental

# of Classes	Optimal Classification	GVF
1	$[220, 500, 1500, 1700, 3480, 7600]$	0
2	$[220, 500, 1500, 1700, 3480], [7600]$	0.826
3	$[220, 500, 1500, 1700], [3480], [7600]$	0.958
4	$[220, 500], [1500, 1700], [3480], [7600]$	0.998
5	$[220], [500], [1500, 1700], [3480], [7600]$	0.999
6	$[220], [500], [1500], [1700], [3480], [7600]$	1

Table 5.7: *Jenks* classification on Q, with calculated GVF

storage capacity when it comes to short-term memory. Miller [35] displayed evidence that people can remember about seven chucks in short-term memory tasks. Later, Cowan [36] suggested that seven is only a rough estimate, and that the limit is actually closer to four chunks. There is a clear limit on this short term memory capacity a human has. Therefore, the quality of the type of analysis presented in this thesis is influenced by the number of classes an analyst gets exposed to at a single time.

To summarize, we want *Jenks* to have enough freedom to discover as many peaks as possible, while simultaneously limiting the number of classes generated by *Jenks* to consider for human limitations. To be able to discover which threshold satisfies both these conditions, extensive testing on BPI 2012, 2014, 2015, 2017, 2018 and 2019 [25, 26, 27, 22, 24, 28] as well as internal ProcessGold datasets has been done. Table 5.9 shows the number of classes generated by *Jenks* when varying the *RGVFC_threshold*, per dataset (non-aggregated test results can be found in Appendix A).

These tests show us a number of things. First, when looking at the average number of classes generated, we see that a threshold of anything higher than 0.5% results in the number of classes generated falling within the mental storage capacity limit of four to seven items, as suggested by Miller and Cowan [35, 36]. Furthermore, it gives enough freedom to *Jenks* to generate at least 6 classes (if so many peaks are present within the data). However, when looking at the distributions of the number of classes generated in Appendix A, when setting the threshold at 0.5%, significant portions of segments are classified into more than five/six groups. This threshold causes around 20 to 40 percent of segments to be classified into more than seven classes. The chance that segments get classified into more than seven groups is too high, since we still have no knowledge on the form of multimodal distributions found in event logs. We like to err on the side of caution, therefore, a safer option is a threshold in the range of between 2.5%, and 10%, since this range will result in a maximum of five to seven classes. We suggest that setting the threshold within this range allows *Jenks* enough room to detect (at least) the most obvious peaks, while simultaneously keeping the number of classes low enough to allow for proper analysis by an analyst.

Limitations of the RGVFC_threshold It is important to note that this approach is vulnerable to localized maximums. When taking an arbitrary set of numbers, the change from (for example) three to four classes can be smaller than the *RGVFC_threshold*, but there is no reason a change from four to five classes would not a RGVFC that is greater than the *RGVFC_threshold*. However, we have not encountered a single occurrence of this happening during our testing.

# of Classes	GVF	Increase	Percentage
2	0.826	-	-
3	0.958	+0.132	+15.90%
4	0.998	+0.040	+0.040%
5	0.999	+0.001	+0.001%
6	1	+0.001	+0.001%

Table 5.8: *Jenks* classification on Q, with GVF change

Dataset \ Threshold	0.1%	0.5%	1%	2.5%	5%	10%	25%
BPI2012 (avg)	9.5	6.05	4.98	3.94	3.36	2.88	2.37
BPI2017 (avg)	8.3	5.51	4.62	3.66	3.2	2.69	2.29
BPI2018 (avg)	7.74	5.35	4.46	3.68	3.07	2.65	2.26
BPI2019 (avg)	9.87	6.48	5.33	4.23	3.51	2.96	2.37
PG Internal (avg)	9.56	6.23	5.21	4.12	3.45	2.9	2.32
BPI2012 (max)	19	11	9	7	5	4	3
BPI2017 (max)	18	11	9	7	5	5	3
BPI2018 (max)	20	12	10	7	5	4	3
BPI2019 (max)	16	10	8	7	5	4	3
PG Internal (max)	18	11	9	6	6	4	3

Table 5.9: Number of classes generated by *Jenks* when varying *RGVFC_threshold*

5.5 Segment classification output

Now that we have a method for classifying performance information into distinct groups, we want to demonstrate how our method works on a real event log. Additionally we want to evaluate whether the returned classes from this real event log are meaningful. The following example is based on segment data found in the BPI2017 [22] event log. The segment which we analyze is the segment *W_Call after offers+schedule* $\rightarrow$ *W_Call after offers+withdraw*. Every case in the event log that contains this segment is recorded, which results in 2010 different cases passing through this segment. For every case, the *segment time interval* data was recorded (as explained in Section 2.3). After applying the methods explained in Section 5.4 on the segment performance information, the output consists of four classes. Each class consists of 772, 675, 402 and 161 cases, respectively. In Table 5.10, we see that every class consists of a set of measures (i,e, Min, Max, Median, Average, Size etc.). Due to the nature of *Jenks*, the following statement is always true: $max(Class_{i-1}) < min(Class_i)$ and $max(Class_i) < min(Class_{i+1})$ where i is the class number. In the same table, we can see that the *time intervals* of (cases in) class 1 are, on average, considerably lower than the time intervals of (cases in) class four (i.e. class 1 goes significantly faster through this segment than cases in Class 4). Another way to visualize the classes is using a distribution plot. In Figure 5.6, we show the distribution plot. This figure gives an indication of how the segment *time intervals* are distributed. The vertical blue lines indicate the breaks that *Jenks* computed. We can see that (high) peak in the distribution was captured in a separate class.

This classification already allows for several interesting business questions to be asked, namely:

1. What is the average difference in *time interval*, between quick and slow cases? (Is this difference reflected in our design of the process, or is this unexpected?)
2. What is the ratio of quick to slow cases? (Do our KPIs allow for this many cases to be slow, or do we not regard these cases to be slow at all?)
3. What activity/case attributes makes the quick cases differ from the slow cases? (What specific attributes makes a case slow, can we adapt our process to accommodate for this?)
4. How does performance in a segment, relate to the performance of the entire case? (If a case is slow in a segment, is the throughput time of the entire case also slow?)

Such questions can be explored for every segment in a dataset. Being able to answer these questions can provide significant insight into the workings of a business process, since it allows to analyze a process piece-by-piece. Eventually, our goal is to provide business analysts with a jump start in process analysis by extracting information from them *a priori*, and *automatically*. Having

this information supports identification of bottlenecks (i.e. discovering segments which have a high probability of being a bottleneck). Up until now, our solution does not yet allow for segments to be compared to *each other*. If we want to be able to identify which parts of a process most likely to be bottlenecks automatically, we need to extend our current classification. This problem is expanded upon in Chapter 6. By observing the results of Table 5.10 and Figure 5.6, we demonstrated that we solved **RQ 1:** *Can performance information, extracted from event log segments, be (automatically) subdivided into different performance classes which each represent a part of the variability in performance?*

Class	Median	Avg	Min	Max	stdev	Size
1	2 days 4:30:26	2 days 2:43:00	0:30:16	4 days 0:59:00	1 day 1:35:52	772
2	5 days 3:42:26	5 days 7:14:00	4 days 1:04:41	8 days 0:57:52	1 day 1:37:13	675
3	10 days 3:07:05	10 days 8:32:12	8 days 1:01:50	15 days 1:14:57	1 day 19:48:23	402
4	18 days 5:43:57	19 days 3:46:34	15 days 1:30:55	27 days 1:00:28	3 days 3:39:50	161

Table 5.10: Overview of classes with their performance measures

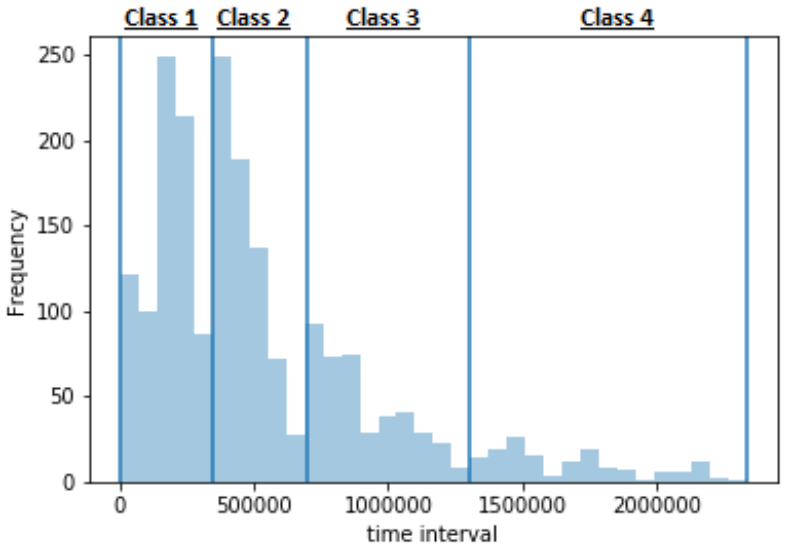

Figure 5.6: Segment *time interval* distribution, with breaks computed by *Jenks*

Chapter 6

Extension, analysis, and classification of performance classes

In Chapter 5, we obtained performance classes from each segment within an event log. We dedicate Chapter 6 to answering **RQ 2:** *Can a set of analyses be created, as to discover similarities/differences within and between the performance classes?*

To answer this question, in Section 6.1, we extend the current *segment* classification by including *overall* performance classes, we explain how *overall* performance classes can be generated from the cases in a segment. In Section 6.2, we formulate several measures of performance classes. For each measure, we explain how and why these measures can be used to identify bottlenecks in segments. Additionally, in the same section, we create algorithmic solutions to obtain these measures from segments automatically. Finally, in Section 6.3, we explain how we create a ranking based on the magnitude of each measure, and use this ranking as starting point of our visualization.

6.1 Overall process performance classes

Ultimately, we want to be able to evaluate every segment within an event log, and determine which segment(s) have the highest probability of being bottlenecks (with respect to the entire process). Currently, we have no way of directly comparing segments to each other. Simply comparing performance and distributions between segments provides no valuable information, because cases in a segment do not necessarily have any overlap with cases from other segments. We can, however, discover how a segment relates to the *overall process* it is part of. Take, for example, the (sub)set of cases presented in Table 6.1. These cases all contain the segment $B \rightarrow C$. To recall, in Chapter 5, we calculate the *segment performance classes*, by determining to which *segment performance class* each case belongs, using *Jenks*, the outcome of this is shown in Figure 6.2. We see that this set of cases was divided into four distinct performance classes. Additionally, we see the distribution of each *segment performance class* using boxplots.

When we combine the traces in Table 6.1, we can see that these combined traces make up a larger process, displayed in Figure 6.1. We can see that a segment is part of a larger *overall process*. We would like to understand how the performance a case has in a segment relates to the performance that same case has *overall*. We can gain insight on the relation of this segment to the entire process relatively simply. To determine the *overall performance classes*, we keep the same division of cases (i.e., Case 1 and 2 belong to class 1, Case 3 belongs to class 2, etc.), but instead of using the *segment time*, we now use the *total time* of each case. Table 6.1 has a column which displays the *total*

Case ID	Trace	Segment time	Segment performance class	Total time
1	A → **B** → **C** → D	10	1	100
2	A → **B** → **C** → F	12	1	110
3	E → **B** → **C** → D	30	2	180
4	E → **B** → **C** → F	80	4	280
...	...	...	...	

Table 6.1: Overview of cases, with their respective traces, segment time, and total time

time of each case. For each class, we gather the total time of each case contained in the class. We make a boxplot of the new overall performance classes, the outcome of which is shown in Figure 6.1.

Discovering how this segment relates to its *overall process* provides insight into a segment on a higher level. It shows what the contribution of that segment is towards the process overall (i.e., "to what degree is this segment responsible for the overall performance of cases?"). In Figure 6.2 and 6.3, we see that both segment classes and the overall classes are increasing, meaning that the average throughput time of a class is larger than its previous class(es). From this observation, we can claim that there is a correlation between performance of a segment, and performance of the overall process. We observe that cases for which performance is slow in this segment, also perform slowly in their overall process, which could be an indicator that this segment causes a bottleneck. Many such correlations can be found between performance classes. Instead of relying on manual analysis of figures such as Figure 6.2 and 6.3, we would like to automate the analysis of performance classes, as to determine what characteristics performance classes have, and how they relate to each other. In Section 6.2, we explore several different statistical measures for determining the characteristics of performance classes.

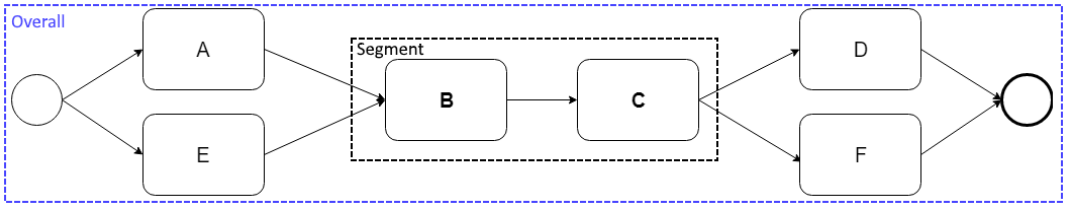

Figure 6.1: *Segment* and *Overall* processes

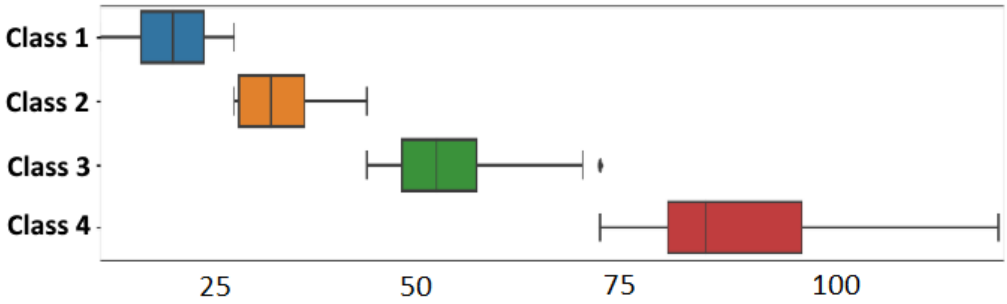

Figure 6.2: *Segment* performance classes

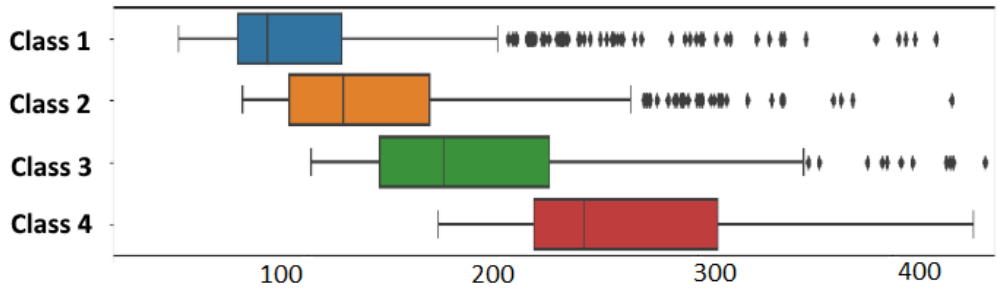

Figure 6.3: *Overall* performance classes

6.2 Analysis of performance classes

In the previous section we expanded our segment classification by adding *overall* performance information. However, analyzing every segment in-depth requires a significant amount of work. Therefore, We would like to know which segments we should focus our process optimization efforts on, by first discovering a set of measures which quantify *segment* characteristics. Additionally, these measures can be used to identify whether a segment contains a bottleneck. We approach this problem by first setting up a list of questions we want to answer:

Measures **within** *segment*/*overall* performance classes:

1. Do the performance classes have significant overlap, or not? (Effect size)
2. What is the time difference between the slowest and fastest classes? (Potential lost time)

Measures **between** *segment*/*overall* performance classes:

3. What is the influence of the segment time, with respect to the total time? (Total weighted impact)

We are looking for methods to find answers to these questions *automatically*. Therefore, sections 6.2.1, 6.2.2, and 6.2.3 are dedicated to finding algorithmic solutions which answer these questions automatically.

6.2.1 Effect size

The purpose of this section is to create a method to answer the question: "Do the classes have significant overlap, or not?". When classes do not have significant overlap, there is a higher chance that a class holds information that is not present in the others. To give an idea of what we mean by that, consider Figures 6.4 and 6.5.

The classes from Figure 6.4 have significantly more overlap in the 25^{th} to 75^{th} percentile than the classes of Figure 6.5. Essentially, Figure 6.4 gives us only rudimentary information (i.e., "There is no difference in throughput time between classes") information. Alternatively, Figure 6.5 shows a clear distinction between every class. This distinction gives reason to analyze the causes for the differences between these classes (i.e. bottleneck analysis). The difference between these classes can be quantified as the *effect size*.

In statistics, *effect size* is a quantitative measure that measures the difference between 2 classes. In this research, we are looking for a non-parametric approach to calculating effect size. A widely used non-parametric effect size technique that tries to quantify the difference between two groups is the $Cohen's\ d$ method [23] (Hereafter denoted as *Cohen's*). This method is chosen because it is non-parametric, works with numeric data, and handles groups that are of different sizes.

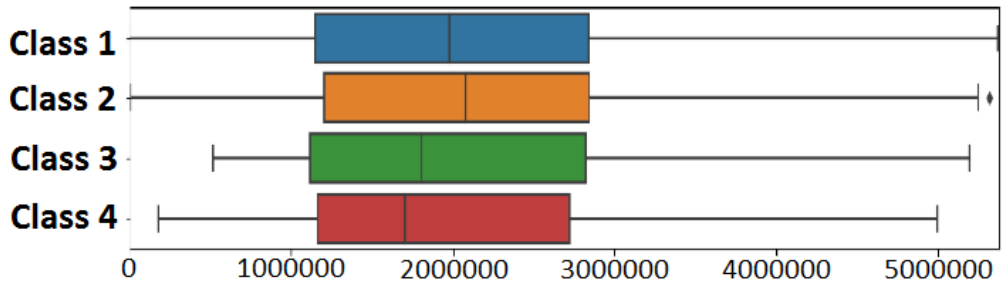

Figure 6.4: Classification with (relatively) small effect size

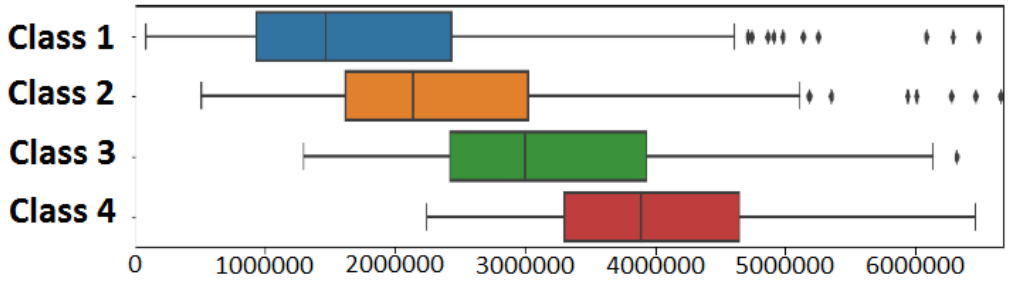

Figure 6.5: Classification with (relatively) large effect size

Cohensd(X_1, X_2) is determined as follows:

$$d = \frac{\bar{x_1} - \bar{x_2}}{s} = \frac{\mu_1 - \mu_2}{s}$$

where s is defined as the pooled standard deviation,
n_1 is the size of X_1, and n_2 is the size of X_2:

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}},$$

where the variance for both groups (s_1^2 and s_2^2) are defined as

$$s_1^2 = \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} (x_{1,i} - \bar{x_1})^2$$

and

$$s_2^2 = \frac{1}{n_2 - 1} \sum_{i=1}^{n_2} (x_{2,i} - \bar{x_2})^2.$$

A *Cohensd* of 1 implies that two groups differ by 1 standard deviation, a d of 2 indicates they differ by 2 standard deviations, and so on. These are equivalent to z-scores.

Cohen's method can only calculate the effect size between two groups at a time. However, we can have more than two classes be discovered by *Jenks*. For our purposes, we calculate an effect size for the *segment/overall* performance classes, by averaging the effect size between every combination of performance classes. To obtain the *average Cohen's d* for *segment* performance classes, or for *overall* performance classes, we apply the following steps:

1. Select *segment/overall* performance classes T, which hold N classes.
2. Find every pair of classes (T_i, T_j) where $i < j$ and $i, j \leq N$

3. Calculate for every pair (T_i, T_j) *Cohen's d*
4. Divide the sum of these by the total number of pairs

Recall in Section 6.1, we determined that correlations exist between the *segment* performance classes, and the *overall* performance classes. Now that we know these correlations exist, we need to determine whether the measures of our performance classes give ground for analysis, by being significantly 'different' from one another. For two groups to become statistically significantly different, Cohen [23] suggested that a d of 0.2 can be considered a 'small effect size', a d of 0.5 can be considered a 'medium effect size' and, a d of 0.8 can be considered a 'large effect size'. These values are expanded upon by Sawilowsky [37], who added, that a d of 1.2 can be considered a 'very large effect size' and a d of 2.0 can be considered a 'huge effect size'

Using this method on the *segment* classification is not useful. We already know that every *Jenks* classification has a high effect size, since it has the property where $max(T_{i-1}) < min(T_i)$ and $max(T_i) < min(T_{i+1})$ where i is the class number. Due to this property, none of the classes have any overlap, which means an effect size which is always significantly larger than 2.

However, the *overall* performance classes do have overlap. This means we are able to measure the effect size of these classes using *Cohensd*. Discovering whether an *overall* performance class is significantly different from the other classes makes that class an interesting candidate for deeper analysis. We want to spend our analysis time efficiently, and therefore need to make a decision what segments to analyze. For the purposes of this research, we decide that performance classes with an *average Cohen's d* of atleast 0.5 (medium effect size) are considered significantly interesting. When a segment has a high effect size, it indicates that a bottleneck might be present in the segment, as well as in the *overall process*. However, a segment can still be interesting if it has a low effect size, if the other measures (detailed in following sections) are substantially high. If this is the case, the segment in question may still be a bottleneck, albeit a local one.

6.2.2 Potential lost time

The purpose of this section is to create a method to answer the question: "What is the time difference between the slowest and the faster classes?". We calculate the Potential lost time, using only the *segment* performance classes. Potential lost time is calculated as follows:

1. For every class C_i in the *segment* performance classes, where i is the class number, calculate $median(C_i)$
2. Take $median(C_1)$ as $optimal\_time$
   (Due to the nature of *Jenks*, we know Class 1 always has the lowest (i.e., fastest) median)
3. For every class C_i, calculate $C_i\_lost\_time = (median(C_i) - optimaltime) * size(C_i)$
4. Calculate $\sum_{i=1}^{N} C_i\_lost\_time$, where N is the total number of *segment* performance classes.

It is important to note that we have classes which are not necessarily of equal size. This means that we cannot simply take the sum of *time intervals* of Class 1 (fastest class), and compare that to the other classes, to calculate *lost time*. Instead, we use the *median* of each class. We use the *median* for two reasons: First, due to the nature of *Jenks* (discussed in Section 5.4), we know the that the $median(C_1)$ will always be the lowest median compared to the other classes. Second, we use the *median* of a class, because it is less skewed by imbalanced data, than using the *average* of a class.

Using this method, we can calculate the *potential lost time* for each segment. This measure depicts the *magnitude* at which cases are slower *than the fastest possible time in that segment.* It provides an answer to the question; "What is the highest (theoretical) performance gain our process can have, when using only information from the event log?" This measure is designed to gain general

Class	Size	Segment time median	Optimal time	Potential lost time
A1	1000	10	10	$1000 * (10 - 10) = 0$
A2	800	40	10	$800 * (40 - 10) = 2400$
A3	400	120	10	$400 * (120 - 10) = 44000$
A4	200	3000	10	$200 * (3000 - 10) = 598000$
				644400
B1	1000	500	500	$1000 * (500 - 500) = 0$
B2	800	1000	500	$800 * (1000 - 500) = 400000$
B3	400	1500	500	$400 * (1500 - 500) = 400000$
B4	200	1800	500	$200 * (1800 - 500) = 260000$
				1060000

Table 6.2: Lost time calculations for segments A and B

insight into the segment performance of an event log, without having knowledge on the specific details of the business process. It also provides a guideline for deciding whether the segment is worth improving. If segment A has a relative small lost time compared to another segment B, this indicates that it is more profitable to focus on improving the latter segment.

To give an impression on the calculation of the *Potential lost time*, we present segment A and B, both classified using *Jenks* into classes $A1, A2, A3$, and $A4$ and $B1, B2, B3$, and $B4$, respectively, in Table 6.2. Assume all cases go over both segment A and segment B. Therefore, both the sizes of each class and the overall performance is equal in both segments. Where they differ is the performance difference between their performance classes. In the example, we see that the *lost time* of segment B is higher than the *lost time* of segment A. This gives an indication that, if we have to choose between focusing improvement efforts on segment A or B, focusing on segment B first may be more beneficial since there is theoretically more performance improvement to gain.

6.2.3 Total weighted impact

The purpose of this section is to create a method to answer the question: "What is the influence of the segment time, with respect to the total time?". First, this measure determines the magnitude of impact a segment has on its *overall* performance. Additionally, the measure can be used to determine where improvement efforts should be focused on, when a company wants to improve their *overall* process throughput, but only has the time and resources to improve parts of the process. To make sure that the company focuses their improvement efforts efficiently, it is helpful if they are able to recognize whether a segment is worth improving or not (the least amount of effort for maximal amount of gain).

We introduce a new measure, named *Total Weighted Impact* (TWI). The measure calculates what percentage of the overall performance (of cases) is caused by the segment performance (of cases). We define the *Total weighted impact* for segment S as follows:

$$TWI(S) = \sum_{i=1}^{C} \frac{SP(S_i)}{OP(S_i)} \frac{|S_i|}{|S|}$$

where

$$C = \text{Number of classes in segment S},$$
$$SP(S_i) = \textit{Segment performance} \text{ of segment } S_i,$$

and

$$OP(S_i) = \textit{Overall performance} \text{ of segment } S_i.$$

To give an impression on the calculation of the *Total weighted impact*, we present segment A and B, both classified using *Jenks* into classes $A1, A2, A3$, and $A4$ and $B1, B2, B3$, and $B4$, respectively, in Table 6.3.

Assume all cases go over both segment A and segment B. Therefore, both the sizes of each class and the overall performance is equal in both segments. Where they differ is the performance of the segment itself. In segment A, cases in class $A1$ are, on average, 300 times faster than cases in class $A4$. On first sight, this would mean that this segment is a good candidate to focus improvement efforts on (Section 6.2.2 explained how *lost time* is calculated). However, we can also see that that particular segment is only a fraction of the *overall* process, since it only accounts for around 6.25% of the total throughput time. While focusing improvement efforts on this segment may improve local performance, it does not have a significant impact on the overall process (i.e., while there may be a bottleneck in the segment itself, the slow performance in the *overall* process could be caused by another segment in the same *overall* process). Alternatively, in the same table, we see that segment performance in B is a significantly large portion of the overall performance (41.15% in total). This means that focusing our improvement efforts on this segment will not only benefit local performance, but *overall* performance too.

Class	Size	Segment Performance	Overall Performance	Weighted Impact
A1	1000	10 minutes	1000 minutes	0.42%
A2	800	40 minutes	4000 minutes	0.34%
A3	400	120 minutes	3000 minutes	0.67%
A4	200	3000 minutes	4000 minutes	6.25%
				7.68%
B1	1000	500 minutes	1000 minutes	20.80%
B2	800	1000 minutes	4000 minutes	8.30%
B3	400	1500 minutes	3000 minutes	8.30%
B4	200	1800 minutes	4000 minutes	3.75%
				41.15%

Table 6.3: TWI calculations for segments A and B

6.3 Ranking of segments

In the previous sections, we discussed several methods for measuring different aspects of a performance classes. Every aspect contributes to deciding whether a segment holds a potential bottleneck, and therefore is worth looking at in-depth. To give a quick overview, we first recall what the measures are, and what they do.

1. Effect size (Section 6.2.1), quantifies the average overlap performance classes have.

2. Lost time (Section 6.2.2), quantifies how much the throughput time of a segment could (potentially) be improved.
3. Total weighted impact (Section 6.2.3), quantifies the impact of segment performance, on the overall throughput time.

These measures cannot be compared directly, because they quantify different aspects of a segment. When a segment has a high *lost time* and a low *effect size*, it does not mean that the segment is more/less likely to be a bottleneck compared to a segment which has a low *lost time* and a high *effect size*. However, the measures can be used to form an opinion on whether a segment is more interesting to analyze *relative to other segments*. We do this by creating a ranking based on the magnitude of each of the measures. To provide an understanding of how this ranking works, we provide the following example. In this example we analyze a synthetic event log, with the flow model displayed in Figure 6.6.

We run every segment through our algorithm, and display the results in Table 6.4. Subsequently, we compare the value of each measure to each other, and develop a ranking. Segment $A \rightarrow B$ has the highest *effect size* of 1.44, so this segment will get the highest ranking number of 7 (in this example, there are 7 segments in total), with respect to the effect size. Segment $C \rightarrow D$ has the second highest *effect size* of 1.21, so this segment will get the second highest ranking number of 6, etc. For *lost time* and *weighted impact*, we apply the same rules, resulting in a ranking, also displayed in Table 6.4.

Now that we have ranked each measure individually, we can calculate a cumulative ranking for each segment. This cumulative ranking is simply a summation of the ranking of the three measures. The cumulative ranking determines how segments are more/less likely to contain bottlenecks, relative to each other, i.e., the segment with the highest cumulative ranking is the most likely to contain a bottleneck, and the segment with the lowest cumulative ranking is the least likely to contain a bottleneck.

In this chapter, we extended the *segment* performance classes, with *overall* performance classes. Additionally, we created several measures, each of which quantify different characteristics of performance classes. By observing the results of Section 6.2.1, 6.2.2, and 6.2.3, we demonstrated that we solved **RQ 2:** *Can a set of analyses be created, as to discover similarities/differences within and between the performance classes?*. The ranking was created to give a first direction for identifying bottlenecks. Now, we need to develop a way to present these findings to an analyst in a complete and comprehensible way. This is addressed in chapter 7.

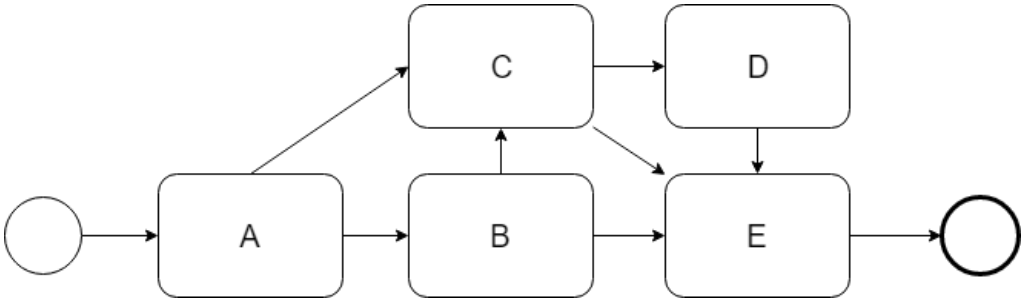

Figure 6.6: Synthetic log flow model

Segment	**Effect size**	**Lost time**	**Weighted impact**	**Effect size ranking**	**Lost time ranking**	**Weighted impact ranking**	**Cumulative ranking**
$A \rightarrow B$	1.44	10000	82%	7	7	6	20
$B \rightarrow C$	0.59	400	85%	2	2	7	11
$C \rightarrow D$	1.21	8000	13%	6	1	2	9
$A \rightarrow C$	0.95	2000	4%	5	6	1	12
$C \rightarrow E$	0.51	500	45%	1	3	4	8
$D \rightarrow E$	0.73	250	20%	3	1	3	7
$B \rightarrow E$	0.85	4000	69%	4	5	5	14

Table 6.4: Example ranking of segments for synthetic event log

Chapter 7

Visualization in the ProcessGold Platform

In Chapter 5, we obtained performance classes from each segment within event logs. In Chapter 6, we provided a measure-based ranking for each of these segments. We dedicate Chapter 7 to answering **RQ 3:** *Can the outcome of the performance class analysis be aggregated and visualized in such a way, as to support bottleneck identification?*

To answer this question, we propose several dashboards that each visualize different parts of the process mining perspectives. In Section 7.1 and 7.2, we address the *time perspective* of process mining by visualizing the outcome of our *segment analysis*. In Section 7.3, we address the *case perspective* by implementing and visualizing case attribute frequency analysis for performance classes. In Section 7.4, we address the *organization perspective* by visualizing event attribute distributions, for different segment types. Finally, in Section 7.5, we address the *control-flow perspective* by visualizing and comparing process flows between classes.

7.1 Segment Overview

We start our visualization by visualizing the *time perspective*. The *time perspective* is concerned with the timing and frequency of events [6]. Timing information can be used to discover (performance) bottlenecks, measure service levels, etc. From the limitations explained in Section 3.1, we can see that our approach is essentially a new way of representing performance related information. Therefore, to visualize the *time perspective*, we present the user (i.e. business analyst) with the outcome of our segment analysis described in Chapter 6.

First, we present an overview of each segment, and its measures, to the user. The first dashboard a user will see is the *Segment Overview* dashboard shown in Figure 7.1. This dashboard has two regions. Region (1) shows an overview of each segment, and its attributes. In Figure 7.2, we zoom in on the first row of the overview. From left to right, the row consists of the following attributes: (a) Name of the segment, (b) Number of cases in that segment, (c) Number of classes in that segment, (d) Effect size ranking, (e) Lost time ranking, (f) Weighted impact ranking, (g) Total cumulative ranking, and finally (h) Inspection Button. The ordering of this overview is based on the arguments made in Section 6.3, where the segment with the highest *Total ranking* is placed at the top. The user is able to sort based on each of the attributes (a) to (g). Additionally, when hovering over (h), the user is shown the real numeric value of the measure, on which the ranking is based. In region (2), we show an information symbol. When the user hovers over the i symbol, they will see the popup shown in Figure 7.3. This popup explains how each segment measure is calculated.

The user can drill-down into deeper analysis of a segment by clicking on the name of the segment. The application will navigate to the next dashboard named Segment Information. This dashboard is elaborated upon in the next section.

Figure 7.1: Dashboard which shows Segments Overview. Region (1) shows a table with an overview of each segment, including their attributes. Region (2) shows the buttons which (when hovered over) explain the segment measures.

Figure 7.2: First row of segment overview

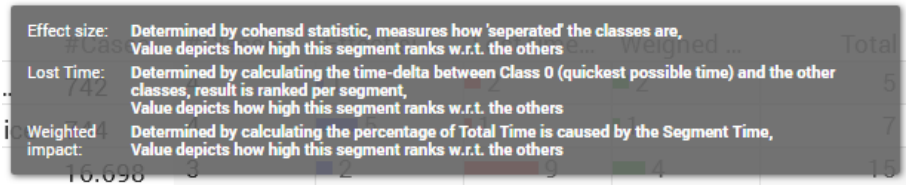

Figure 7.3: Measure information popup

7.2 Segment Information

In Figure 7.4, we show the dashboard *Segment Information*. This dashboard is the continuation of the dashboard described in Section 7.1. This dashboard shows the user an in-depth view of a segment. In this particular example, the segment we are inspecting the segment *Final check of invoice* $\rightarrow$ *Final check of invoice*. Furthermore, the segment consists of three classes, where class 1, 2, and 3 contain 9797, 2871, and 305 cases, respectively. This dashboard consists of four regions.

In region (1), we show a selector, where the user can see which segment they are currently inspecting. Additionally, the user can choose to select a different segment to inspect, if they wish to do so. This allows the user to quickly navigate from the analysis of one segment, to the next.

In region (2), we show an overview of the segment the user is currently inspecting. The overview shows a stacked bar, where each section represents the value of each measure. This gives the user the ability to quickly check how this segment was ranked, relative to other segments. This allows the user to gain general insight on the segment. Additionally, it gives the user an idea of where to

start their analysis. For example, if *effect size ranking* is high, and the *lost time ranking* is low, it may be interesting to analyze the distributions presented in region (3) and (4) in-depth, to get insight into reason why.

In region (3), we show the distribution of the total throughput time for each case, per class. It essentially shows the distribution of each class in the *overall performance class type*. This region is designed to allow the user to compare the classes visually, which creates an understanding of the *overall performance* of each class.

In region (4), we show the *median* throughput time of each class, and how this throughput time is subdivided. Additionally, we show a subdivision of the cases. This subdivision consists of three parts: Region (4a) shows Elapsed time (from the start of a case until the segment is reached), Region (4b) shows Segment time (time the case spent in the segment), and Region (4a) shows Remaining time (time the case spent between exiting the segment, and the end of the case). In this particular example, the elapsed time is 23 hours, the segment time is 2.6 hours, and the remaining time is 4 days. This region essentially visualizes the *weighted impact factor* measure, as it shows how the *segment performance* relates to the *overall performance*. This provides the user with insight on the impact a segment has has, on the process as a whole. Additionally, each column shows the performance of a class, relative to other classes. This allows the analyst to visually compare classes, and create insights on the relative performance of each class.

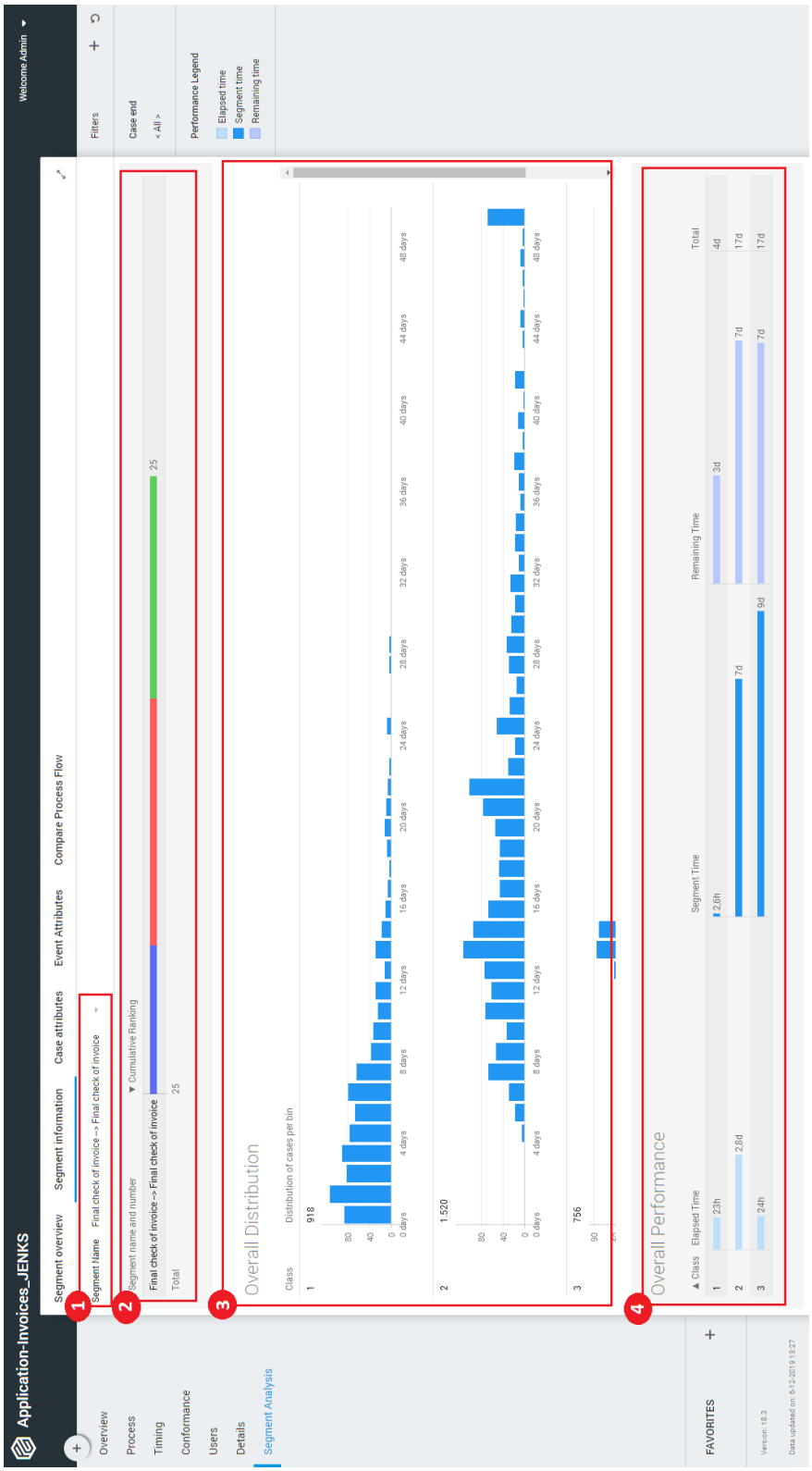

Figure 7.4: Dashboard which shows Segment Information - Region (1) shows the segment selector. Region (2) shows the measures of a segment. Region (3) shows the distribution of case throughput times. Region (4) shows a distribution of Elapsed (4a), Segment (4b), and Remaining time (4c), for each class

7.3 Case Attributes

We show the Case Attribute dashboard in Figure 7.6. This dashboard is dedicated to the analysis of case attributes, present within our segments. The reasoning behind this dashboard are as follows. First, it provides the user with the ability to compare the *case attributes* of different classes. It allows the user to, for example, see if there are discrepancies between the *case attributes* of slowest performance class, compared to the fastest *performance classes*. The *case attributes* may reveal *why* there is discrepancy in performance.

However, the *performance classes* are not necessarily of equal size. Also, the distribution of attributes over a set of cases is always relative to the size of the segment (i.e., 40% of cases in this class contain the *case attribute A*). Due to these factors, it is difficult for a user to judge whether a class contains *significantly* deviating *attribute* frequencies, compared to other classes. This is where the power of *frequency analysis* [15] comes into play. The *frequency analysis* provides a objective **statistical** basis for judging whether the attributes frequency of a class is *significantly* deviates from other classes.

In region (1), we show two selectors. The user can use selector *Segment Name* to select which segment the user wants to analyze. Additionally, the user can use selector *Select Class* to select which (set of) class(es) they want to analyze.

In region (2), we show two thing, first, selector *Case Attribute*, where the user can select which case attribute they want to perform frequency analyses on. Additionally, a button *Only show significant*, which, when enabled, only shows those attributes that are significantly different than expected, based on research done by Verhoef [15].

In region (3), we show a chart, where the black line denotes what the expected frequency of the selected case attribute. The blue bar shows what the observed frequency of the selected case attribute is.

In region (4), we show a table, which depicts how the selected *Case attribute* is distributed over all classes. In the same region, the user can use the *Percentage/Count* selector to switch between displaying the distribution as percentage, or as number.

In this particular example, we have selected the segment 'Checked and approved → Pay invoice', and class 1. Additionally, we have selected 'Case Type' to be the attribute we want to perform *frequency analysis* on. We can see in region (3) of Figure 7.6 that case 'Case Type' has four variants. Namely, Partner, Small invoice, Medium invoice, and Preferred supplier. After enabling the *Only show significant* button, we can see (in Figure 7.5) that the attribute 'Partner' which appears statistically more frequently in class 1 (61% instead of 43.24%) than expected. Additionally, attribute 'Preferred supplier' appears statistically less frequently in class 1 (25% instead of 43.24%) than expected.

Figure 7.5: Frequency analysis, only showing significantly different attribute frequencies

Figure 7.6: Dashboard which shows Frequency Analysis - Region (1) shows the segment selector. Region (2) shows which case attribute the user is currently inspecting, including a button which activates significance filtering. Region (3) shows the expected and observed frequencies of case attributes. Region (4) shows the distribution of the case attribute over all classes

7.4 Event Attributes

We show the *Event attributes* dashboard in Figure 7.4. This dashboard is dedicated to visualizing the *organizational perspective* of process mining. The *organization perspective* focuses on resource, i.e., which actors (e.g., people, systems, roles, and departments) are involved and how are they related. Nearly all process mining tools consider resource information as plain data elements [6]. Therefore, we can visualize this perspective by providing the analyst with tools to analyze case and event attributes. We already created a visualization for analyzing case attributes in Section 7.3. We extend analysis by adding a visualization for event attributes in Section 7.4. We do this by adding a dashboard that allows for the comparison of event attribute frequency between classes. In region (1), we show a selector, where the user can see which segment they are currently inspecting. Additionally, the user can choose to select a different segment to inspect, if they wish to do so. This allows the user to quickly navigate from the analysis of one segment, to the next.

In region (2), we show two selectors, the first of which is *Event Attribute*. Using this selector, the user can select which event attribute they want to analyze. The second selector is *Percentage/-Count*, where the user can select whether to show distribution of values a percentage, or as a total count of events. This allows for flexibility in analyzing frequencies.

In region (3), we show a table, where the distribution of the selected event attribute is shown, per class, based on the segment activity. If 'percentage' is selected, every row sums to 100%. This allows the user to inspect the event *attribute frequencies* of the *segment process*, and allows them to compare these between different classes.

Region (4), is functionally the same as region (3), albeit using the event attribute frequencies of the *overall process*, not just the segment itself. This allows the user to inspect the frequencies of the *event attributes* in the *overall process*, allows them to compare these between different classes.

In this particular example, we have selected the segment Check received invoice → Final check of invoice. We have selected 'Country' as event attribute to inspect. We see in region (3) of 7.7 that this event attribute has 7 variants, namely 'Austria', 'France', 'Germany', 'Netherlands', Switzerland', 'UK', and 'US'. Additionally, we see that 'Austria' seems to be frequently present when in the slower classes 4 and 5. Therefore, in this example, 'Austria' could be an identifier for performance bottlenecks in this dataset. Finally, region (4) shows the distribution of *event attributes* in the *overall process* associated with this segment.

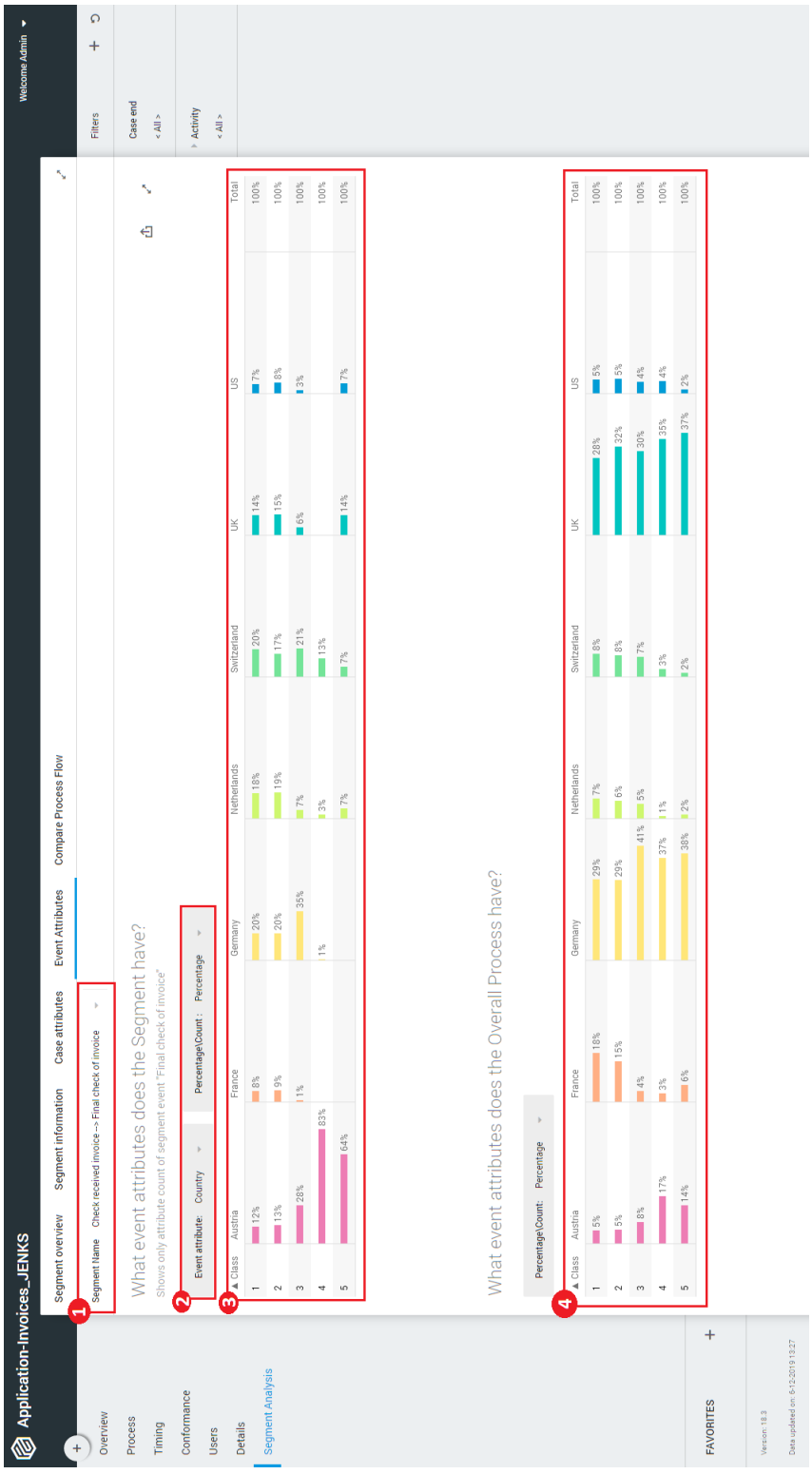

Figure 7.7: Dashboard which shows Event Attributes - Region (1) shows the segment selector. Region (2) shows which event attribute the user is currently inspecting, also a selector where the user can switch between percentage and counts. Region (3) shows the distribution of event attributes in the *segment*. Region (4) shows the distribution of event attributes in the *overall* process

7.5 Process Flow Analysis

Finally, we show the *Compare Process Flow* dashboard in Figure 7.8. This dashboard is dedicated to visualizing the *control-flow perspective*. More specifically, this dashboard can be used to discover if the are differences between the flow of one class, with respect to other classes. The *control-flow perspective* focuses on the control-flow, i.e., the ordering of activities [6]. The goal of this perspective is to get an understanding of the process-flow a process. We want to compare the process-flow of one class, directly to other classes. This assists in bottleneck identification by allowing analysis between, for example, the fastest class and the slowest class. Significant deviations in process-flow could be the cause of performance loss. ProcessGold already provides process-flow analysis capabilities in the form of a process graph viewer, as discussed in Section 2.5. Therefore, we extended the existing process viewer to work with segment classes, and develop a dashboard accordingly. The dashboard consists of several regions.

In region (1), we show a selector, where the user can see which segment they are currently inspecting. Additionally, the user can choose to select a different segment to inspect, if they wish to do so. This allows the user to quickly navigate from the analysis of one segment, to the next.

In region (2a,b and c), we show a process flow analysis tool, in region (2a), we show a selector *Process A*, where the user can select which class(es) to inspect. In region (2b), we show the process flow of the selected class(es). Additionally, every edge show what percentage of cases that traverse it. In region (2c), we show the legend which explains the process utilization rate, number of cases in the flow, and has user adjustable slides, which sets the threshold for hiding/showing infrequent activities and edges.

In region (3a, b, and c), we also show process flow analysis tool, This region is functionally the same as region (2). The only difference is that it shows the classes selected in region (3a) *Process B*, instead of the classes selected in region (2a) *Process A*.

In region (4), we show two buttons, where the user can select between showing the process as a *Side-by-Side* view, or a *Combined*. The *Combined* view is shown in Figure 7.9. Here, the application shows combines the cases selected in *Process A* and *Process B*, and builds a new process flow. The orange numbers and activities are associated with *Process A*, and the green numbers and activities are associated with *Process B*. In this view, it is possible to view additional statistics on the edges, such as average throughput times, which is shown in Figure 7.10.

In this particular example, we have selected the segment *Final check of invoice → Final check of invoice*. Additionally, we have selected Class 1 as input for *Process A*, and Class 2 + 3 for *Process B*. If we inspect the combined view in Figure 7.9 we can see that for most edges, the percentage of cases that traverse the edge are equal. However, the edge between *Check received invoice* and *Final check invoice* show that the edge traversal rate of *Process A* is significantly higher than that of *Process B* (99% vs 53%). Instead, a significant portion of the cases in *Process B* execute the activities *Request data* and *Check contract conditions*. Remember that Class 1 contains the cases that passed the segment quickest. Therefore, in this case, traversing *Request data* has significant impact on the performance of the segment *Final check of invoice → Final check of invoice*. Users can use this view to discover whether there are discrepancies between the flows of quicker classes with respect to slower classes, which in turn supports root cause analysis.

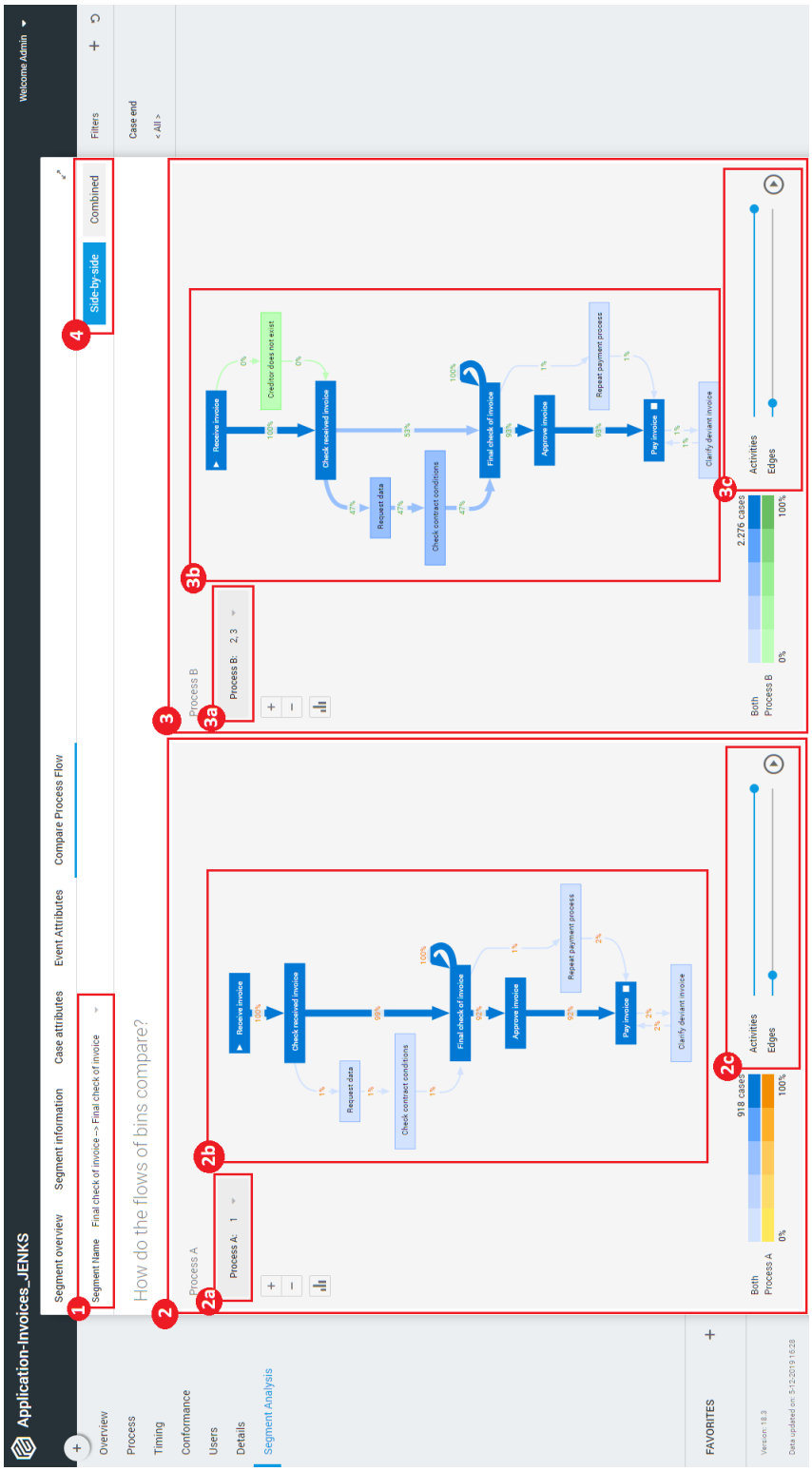

Figure 7.8: Dashboard which shows Compare Process Flow - Region (1) shows the segment selector. Region (2) shows the process flow analysis of the classes selected in region (2a). Region (3) shows the process flow analysis of the classes selected in region (3a). Region (4) shows two buttons, which the user can use to switch between Side-by-side and Combined views

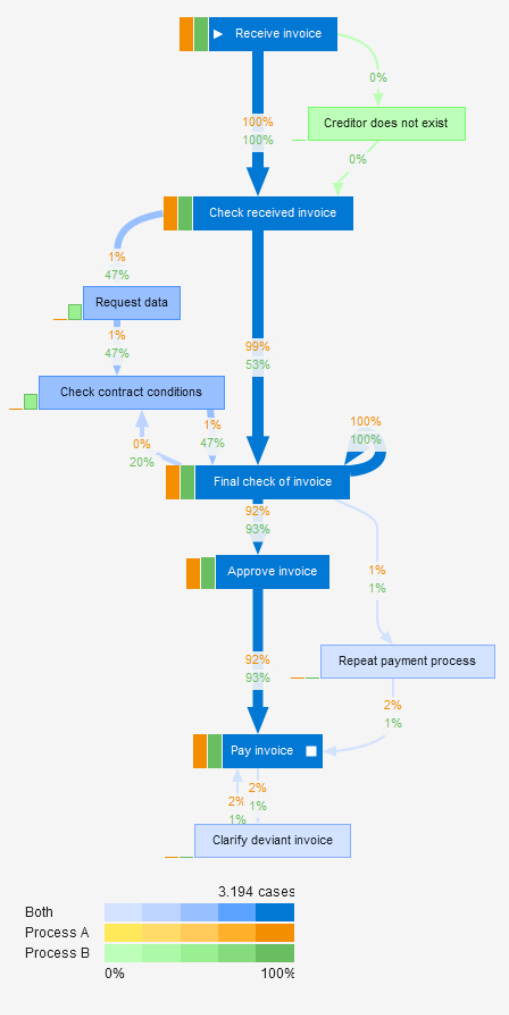

Figure 7.9: Compare Process Flow - Combined - Edge traversal rate

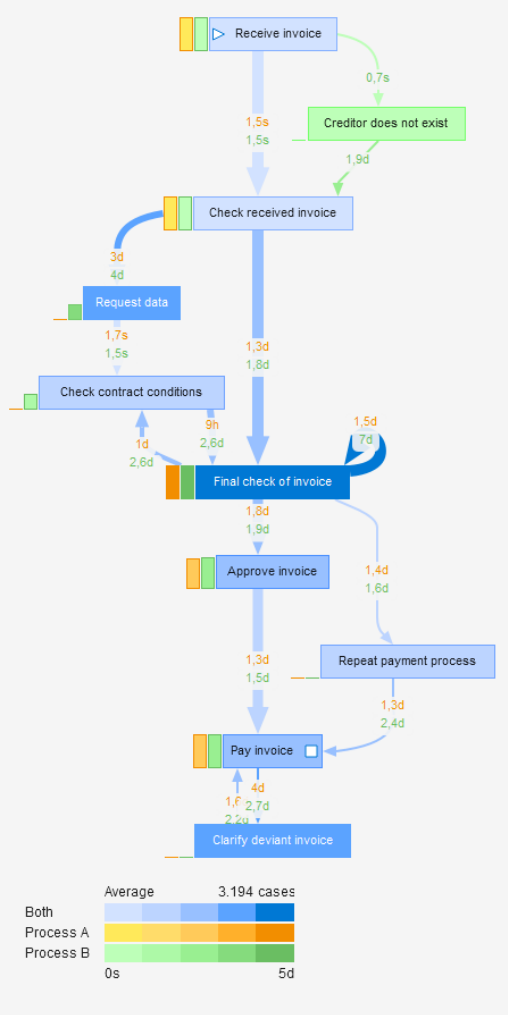

Figure 7.10: Compare Process Flow - Combined - Average throughput time

Chapter 8

Evaluation

The main goal of this research is to support bottleneck identification by automating analysis of previously unexplored parts of process mining, namely, *segment analysis*. In Chapter 7, we proposed several dashboards which each visualize a different perspective of process mining. In this chapter, we explore the understandability, usefulness, and correctness of our *segment analysis*, and how it supports bottleneck identification through these dashboard. To do this, we set up two distinct evaluations of our developed techniques. In Section 8.1, we evaluate the correctness of our approach, by applying it on BPIC datasets, and comparing our findings, with those of the BPIC reports. In Section 8.2, we evaluate the understandability and usefulness of our approach by performing validation tests with ProcessGold analysts. These tests also show whether it is possible for analysts to analyze an event log, quicker, and more thorough, compared to when they use contemporary process mining tools.

8.1 BPI Challenge and Report

The International Conference on Business Process Management (BPM) [16] is the premier conference for researchers and practitioners in the field of Business Process Management and by extension, process mining. For these conferences, Business Process Intelligence Challenges (BPIC) are created. These challenges consist of real life event logs, which participants are allowed to analyze for several weeks. These participants can choose to focus on a specific aspect of interest and analyze this in great detail. Examples include the creation of control-flow models, performance models, predictive models, etc. Alternatively, participants can choose to focus on a broader range of subjects, albeit in less detail. The participants have a set amount of time to analyze the event log, whereafter they hand in a report containing their findings. A jury consisting of field experts judge the reports on correctness, usefulness and completeness of analysis. Finally, one or more reports are selected to be the winner of that BPIC. For our evaluation, we use BPIC 2017 [22] dataset.

In this thesis, we evaluate whether an analyst using our approach is able to discover similar *points of interest* (e.g., bottlenecks) as those detailed in the reports of the BPICs. The challenges are generally taken on by process analysts and academics, where each teams consists of upwards of five participants. Compared to the teams who participate in the BPICs, we have far less time, resources, and domain knowledge to use for analysis. Therefore, it is reasonable to expect that our analysis will not be as thorough as the BPIC submissions. To offset this discrepancy, we will use the conclusions of the winning submissions, and analyze whether we come to similar conclusions when using our approach.

Limitations of evaluation through BPIC It is very important to note however, that the BPIC reports mainly focus on process discovery, understanding, and analysis. The reports focus

on discovering trends, distributions, and irregularities in the event logs *as a whole*. This is significantly different from our approach, which mainly focuses on the discovery of performance related bottlenecks, by providing analysts with a preprocessed *segment analysis*, and tools to explore this analysis. Additionally, our approach analyzes smaller subsets of event logs (segments) and bases analysis on these subsets. We try to overcome this difference by comparing our approach only to performance related discoveries described the reports. Additionally, it is important to note that we lack the necessary domain knowledge to fully understand the processes described in the BPIC reports. Therefore, we are not be able to declare that we found a bottleneck, we can only speculate that certain segments, activities, or attributes *show signs* of causing bottlenecks.

8.1.1 BPIC 2017

BPIC 2017 consist of an event log provided by a Dutch financial institution. The event log contains data on a loan application process, which has seen an significant increase in cases since the financial crisis of 2008. The event log consists of 31,509 cases, 1,202,267 events, 26 activities[1] and 149 users. Additionally, the log contains various non-standard case and event attributes, such as, *case outcome*, *Loan reason*, *Application type* and *Customer type*. The submission consisted of three categories: Student, Professional and Academic.

The Financial institution was particularly interested in answering the following questions:

1. What are the throughput times per part of the process?
2. What is the influence on the frequency of incompleteness to the final outcome?
3. How many clients ask for more than one offer?

For this evaluation, we compare our approach to the winners of the professional and academic categories. In the professional category, the winners were analysts from KPMG Technology Advisory, Belgium [38]. In the academic category, the winners were analysts from Pontifícia Universidade [39].

Preprocessing:

The BPIC 2017 dataset has lifecycle information in it. The lifecycles are *Start*, *Suspended*, *Resume*, *Ate_abort*, *Withdraw*, and *Complete*. As stated Section 5.1, this information can be used to enrich the event log with more information. Normally, we recommend this step, as it allows an analyst to see whether a segment counts as *waiting time* or *processing time*, which adds valuable process information. However, when examining the winning reports of BPIC 2017, we noticed two things. First, the report from Pontifícia Universidade [39] do not contain any analysis based on the lifecycles, and their findings reflect that. Secondly, the report from KPMG Technology Advisory [38] does utilize the lifecycle information, both with respect to *process discovery* (i.e. discovering process models) and when describing their findings when doing a *time analysis*. Therefore, we **do not** use a converted dataset which uses lifecycle information, when comparing our findings to the report of Pontifícia Universidade. The base event log still has a *Case ID*, *activity*, and *timestamp*, and is, therefore, still usable. Alternatively, we **do** use a converted dataset which uses lifecycle information, when comparing our findings to the report of KPMG Technology Advisory.

Additionally, it is important to note that our analysis changes slightly based on the value of user parameters. We tweak these parameters on the type of dataset we encounter, to allow for better analysis. The parameters we adjust are as follows:

1. (*Minimum segment frequency = 1000*), this parameter denotes the number of times a segment must occur in a dataset, before it is selected for analysis. The motivation behind this

[1] 66 activities if lifecycles are converted

▼ Segment name	#Cases	Classes	Effect size ranking	Lost time ranking	Weighed impact ...	Total
W_Call after offers+suspend –> A_Cancelled+complete	7.573	2	48	48	48	144
W_Call after offers+suspend –> W_Call after offers+ate_abort	21.033	3	42	46	46	134
W_Call after offers+schedule –> W_Call after offers+withdraw	2.010	3	47	36	47	130
W_Call incomplete files+suspend –> O_Accepted+complete	4.743	3	41	45	43	129
W_Call after offers+suspend –> O_Create Offer+complete	3.489	3	45	42	42	129
O_Sent (mail and online)+complete –> W_Call after offers+ate_abort	1.384	4	44	38	45	127
W_Call after offers+suspend –> W_Call after offers+resume	24.009	4	35	47	44	126
O_Sent (mail and online)+complete –> W_Call after offers+resume	1.255	4	46	33	41	120
W_Call incomplete files+suspend –> W_Call incomplete files+ate_abort	10.066	4	40	40	38	118
W_Validate application+suspend –> W_Validate application+ate_abort	9.578	3	32	41	37	110

Figure 8.1: Top 10 ranked segments, BPIC 2017 *with* lifecycle information

▼ Segment name	#Cases	Classes	Effect size ranking	Lost time ranking	Weighed impact ranking	Total
W_Call after offers –> A_Cancelled	8.539	2	26	26	26	78
W_Call incomplete files –> O_Accepted	4.783	3	23	25	25	73
W_Call after offers –> O_Create Offer	3.756	3	25	23	23	71
O_Sent (mail and online) –> W_Call after offers	3.272	4	22	22	24	68
W_Complete application –> A_Accepted	22.249	4	24	20	16	60
W_Complete application –> O_Create Offer	1.574	3	20	16	19	55
A_Concept –> W_Complete application	22.264	4	11	24	20	55
W_Validate application –> O_Accepted	11.738	3	12	21	21	54
W_Call incomplete files –> O_Create Offer	1.670	4	21	15	18	54
A_Concept –> A_Accepted	9.231	4	17	18	17	52

Figure 8.2: Top 10 ranked segments, BPIC 2017 *without* lifecycle information

parameter is twofold; First, a dataset with n distinct activities can potentially consist of n^2 different segments. This dataset in particular is quite large, and can theoretically consist of 484 segments. To prevent the number of segments growing too large for an analyst to properly analyze, we use this parameter to limit the number of segments that are eligible for analysis. Secondly, more frequent *segments* can be considered more important, since they are of higher business value/impact. Companies are more likely to focus on high impact improvements.

2. ($RGVFC\_threshold = 7.5\%$), as stated in Section 5.4.2, we suggested selecting a RGVFC_threshold between 2.5% and 10%. After running some tests, we found that a threshold of 7.5% did not overfit (i.e., some classes have more than seven groups), nor underfit (i.e., every class only has two groups) the data, and resulted in every class consisting of two to six classes.

We applied our *segment analysis*, with the above discussed parameters. Figure 8.1 shows the top 10 ranked segments, when analysing the BPIC 2017 dataset with lifecycle information. Additionally, Figure 8.2 shows the top 10 ranked segments, when analysing the BPIC 2017 dataset without lifecycle information. The full tables of ranked segments can be found in Appendix B

Comparison to the report of KPMG Technology Advisory:

Comparison 1: In section 4.2.2, the report [38] shows the average throughput time of each activity, based on the lifecycle information of each activity. Furthermore, the reports determine for each activity, what the item duration (i.e. time that the activity takes in total) and user duration (i.e., time that an employee actively works on the activity) is. In the first and second conclusion of section 4.2.3, the report notes that the activity *W_Call after offers* and *W_Call incomplete files* are very time consuming activities, where there is a large discrepancy between item duration and user duration. This means that most of the duration of the activity is spent on waiting for the client to respond. It is important to note that the analysts had to go through several non-trivial

calculation and filtering steps, before being able to reach these conclusions.

In our *segment analysis* approach, these findings are directly visualized by our ranking displayed in Figure 8.1. In this Figure, we see that the top 3 ranked transition all contain *W_Call after offers*. Segment 1 is *W_Call after offers+suspend* → *A_Cancelled+complete*, which denotes that the client has been called atleast once, but did not accept the offer, either on the phone, or by not responding. Segment 2 is *W_Call after offers+suspend* → *W_Call after offers+ate_abort*, which denotes the that client has been called atleast once, and at some point the client accepted the offer. Segment 3 is *W_Call after offers+suspend* → *W_Call after offers+withdraw*, which denotes the that client was scheduled to be called, but the client accepted the offer before the first call was made. These 3 segments ranking the highest reflect directly the suggestion made by the report, that the financial institution should call the client quicker / more often. Additionally, in the same section, the report states that both the activity *W_Validate application* and *W_Validate application*. Both of these activities appear atleast once in the top 10 in Figure 8.1.

Comparison 2: In the third conclusion of section 4.2.3 *Customer behaviour analysis* [38], the report states, while *assess personal fraud* is very time consuming for cases that have *home improvement* as loan goal. However, this activity only occurs in 1% of cases, and, therefore, does not warrant further analysis. We note that the analysts had to compare processes between different loan goals, and calculate the average total and user duration for each goal. These are several non-trivial data combination, calculation and analysis steps to be able to reach this conclusion

This is directly reflected in our segment overview, by the fact that this activity does not occur in any of the segments we have detected. This is because the parameter *Minimum_segment frequency* filters infrequent behaviour, so this activity was not analyzed due to its relative low occurrence rate. This parameter could easily be adjust based on the wishes of the analyst.

Comparison 3: In the fifth conclusion of section 4.2.3 *Customer behaviour analysis* [38], the report states that the activities such as *handle leads*, *complete application*, and *shortened completion* do not seem to cause problems for the company. We note that the analysts had to compare processes between different loans goals, and calculate the average total and user duration for each goal. These are several non-trivial data combination, calculation and analysis steps to be able to this conclusion

This is directly reflected in our analysis. *Shortened completion* does not occur in any of our segments. Additionally, both *handle leads* and *complete application* are only present in a few segments, and on average rank very low. This is due to them simply not showing any interesting performance characteristics.

Any other conclusions made by the report are not performance based, and we therefore do not compare them to our approach. we conclude that our approach is able discover the same conclusions as the report made, regarding performance. We note that the analysts had to perform several non-trivial analysis steps to obtain these conclusions. Alternatively, using our approach, we were able to reach the same conclusions much faster, because they were discovered automatically using only the first board of our visualization. Furthermore, these conclusions could easily be expanded upon, if the rest of our visualization had been used.

Comparison to the report of Pontifícia Universidade

Comparison 1: In the introduction of the report, the report states the following: "In our diagnostic analysis we show that most bottlenecks are associated with a delay by the applicant to perform an action (e.g., providing documents to the bank)." [39]. According to the details provided by the financial institution, and statements in both winning submissions, activities 'performed by the client' are as follows:

1. **W_Call after Offers**, where the bank worker calls, after the offer is sent, to check whether the client has received it, and asks whether the client is planning to accept it
2. **W_Call incomplete files**, where the bank worker calls to ask for documents
3. **O_Sent (mail and online)**, where the bank send an email with the offer the client, any time after this is the bank waiting for the client to acknowledge that they received to offer

In Figure 8.2, we see a top 10 of the ranked segments of the BPIC 2017 without lifecycles. We see that the four highest ranked segments each contain one of the activities mentioned before. Segments 1 and 3 contain the activity *W_Call after Offers*, this activity means that the bank already called the client, inquiring whether they want to accept the offer. The bank is now waiting for the client to call back with their answer. Segment 2 contains the activity *W_Call incomplete files*, this activity means that the company has inquired the user to provide the correct documentation, and is waiting on the client to provide these. Finally, segment 4 contains *O_Sent (mail and online)+complete*, this activity means that the company just sent the client an offer, and is waiting for them to respond. Every activity which the report denotes as being 'performed by the client' is present in highest ranked segments. Therefore, we can safely conclude the our approach is able to detect the same *performance bottlenecks* as the analysts.

Comparison 2: In section 5.1, the report [38] states: "After the application is created, it passes through the A_Submitted state if it was submitted online, usually by the client itself. As we can see, the client sends most of the applications (20,338 applications, or 64.8%), with a success rate of 50%". The success rate is measured by the percentage of cases reaching activity *A_Pending*. Cases that are created online are cases that pass through the segment *A_Create_Application* → *A_Submitted*.

The same conclusion can be obtain using our approach. If we select the segment *A_Create_Application* → *A_Submitted*, and go to our case attribute board (detailed in Section 7.3), and select *Outcome* as attribute, to table in Figure 8.3 is shown. This figure shows the distribution of attribute *outcome* for every case that goes through *A_Create_Application* → *A_Submitted*, per class. We see that class 1 and 2, 50% of cases have an the outcome *Pending*, class 3 is not counted since it only holds one case.

Additionally, our approach adds valuable performance related information. In Figure 8.3, we see that the distributions of *outcome* are very similar for both class 1 and class 2. Since we know that class 2 is a slower class than class 1, we can conclude that performance has no direct influence on the outcome of a case. This finding is corroborated by the report [38].

Comparison 3: In chapter 6, the report [38] states that in order to perform process analysis, the analysts had to append each duplicated activity with a number, which indicated their relative order. For example, *O_Create Offer1* and *O_Create Offer2*. The analysts base quite a significant part of their analysis on this change. Currently, our *segment analysis* does not include loop depth. So, for example, if a case encounters the segment *A_Create_Application* → *A_Submitted* more than once, only the first occurrences of this segment is counted (The reasoning behind this is explained in Section 9.2. This means there are no distinct activity depths counted. This problem can be overcome by rewriting the event log, and annotate each activity with its respective loop

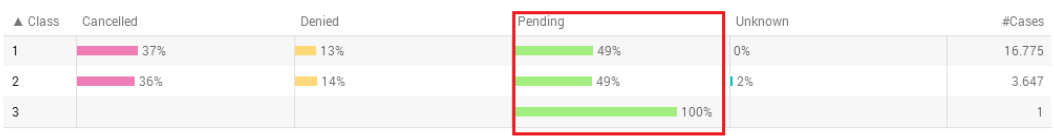

Figure 8.3: Percentage-wise outcome overview for segment *A_Create_Application* → *A_Submitted*

depth. In that case, our *segment analysis* will view *A_Create_Application1* → *A_Submitted1*, and *A_Create_Application2* → *A_Submitted2* as separate entities. In this case, we are confident the conclusions reached by the report can easily be discovered, since most of the analysis of in chapter 6 of the report is superficial.

Any other conclusions made by the report were not performance based, and we therefore do not compare them to our approach. we conclude that our approach is able discover the same conclusions as the report made, regarding performance. We note that the analysts had to perform several non-trivial analysis step to obtain these conclusions, which were not needed when using our approach.

8.2 Validation tests with analysts

The second part of our evaluation are validation tests with ProcessGold analysts/consultants (participants). These tests are designed to gauge the usefulness and understandability of our *segment analysis*, by comparing the techniques and tools participants currently use to identify bottlenecks, to the techniques and tools we have developed in this thesis.

The tests were set up in the following way: Four participants were given two 45-minute tests each. The first test was to analyze a BPIC dataset, the same as they would do with any other client dataset, using the standard ProcessGold AppOne application. The participants were tasked with doing a top-level analysis of the provided dataset, to try and identify (performance) bottlenecks, within a given time frame. If there was time left after the initial identifying step, the participant was tasked to try and find root-causes for the identified bottlenecks. The second test was the same as the first, however, the participants now had to use our *segment analysis* approach, alongside the visualization we discussed in Chapter 7. None of the participants were familiar with the BPIC 2017 and 2018 datasets, and these were therefore chosen to base our tests upon. We divided the datasets over the two tests, making sure that each participant did not see the same dataset twice. So, when a participant used BPIC 2017 for the first test, the second tests was done with BPIC 2018, and vice versa. This resulted in the test division presented in Table 8.1. Each test was recorded, using both sound and video. To understand their thought process while analysing a dataset, we asked each participant to explain each step they took while doing their analysis. The tests were designed to evaluate the following questions:

Question 1: Do the participants understand the *segment analysis* approach?

Question 2: Do the participants understand the visualization?

Question 3: Does the analysis behaviour of the participants change, depending on whether they use our solution, or not?

Question 4: What interesting things (i.e., potential bottlenecks) did the analysts find? Do they find the same bottlenecks when not using our solution?

Question 5: How do participants experience using our *segment analysis*, and our visualiz-

	BPIC 2017 standard	**BPIC 2017 our solution**	**BPIC 2018 standard**	**BPIC 2018 our solution**
Participant 1		X	X	
Participant 2		X	X	
Participant 3	X			X
Participant 4	X			X

Table 8.1: Division of tests, per participant

ation, compared to the standard AppOne application?

Answering question 1 and 2: Before the start of the tests, we gave each participant a 10 minute presentation in which we explained our *segment analysis* approach. Additionally, we spent 5 minutes explaining the visualization. Finally, we spent a few minutes explaining frequency analysis of Section 7.3, since it utilizes a non-standard approach to case attribute analysis. We asked the participant afterwards whether they understood our approach and visualizations. Each participant acknowledged that they understood the general idea of *segment analysis*. Furthermore, that each board of the visualization was self explanatory, and that the frequency analysis was understandable after a few examples were given.

Answering question 3: We did not expect the participants to find deeply hidden bottlenecks, or root-causes within the 45 minute time limit. Proper analysis of processes takes days, if not weeks. However, we did notice a significant difference between the analysis behaviour of the participants when using our approach. The biggest difference was the start of analysis. When using our approach, the participants had a list of potentially interesting segments ranked before they even started. This gave the participants something tangible to start their analysis with. Most participants noted the overwhelming amount of process information they have to go through when using standard process mining tools. Without having a process/domain expert sitting next to them to communicate with, they struggled to get going when using the standard application. It is important to note that the participants were given the option to stop their analysis if they felt like they would not be able to discover new bottlenecks. Out of the four tests performed with the standard application, three decided to stop earlier. Alternatively, every test performed using our segment analysis approach was still going when the 45 minute limit was reached.

Answering question 4: Regarding the BPIC 2017 [22] dataset, every participant that used our solution noted that the case attribute *remaining debt home* appeared significantly more frequently in the slower classes. This conclusion was found on average around five to ten minutes into the test. Alternatively, this conclusion was also found by the analysts using the standard application, however, this was usually around 25 to 30 minutes into the test. This discrepancy had two reasons. First, the analysts spent a significant portion of their time trying to look for interesting things in the *variants* overview of the standard application. However, BPI 2017 has an extremely high number of different *variants*. This resulted in the analysts getting lost in the *variants* view, and finally ending up empty handed. The second reason is that our solution provided the case attribute overview in one simple visualization. This Analysts using the standard application had to first manually combine several attributes into one view, before being able to see the same thing.

Additionally, every participant using our solution also noted that the activity *W_Call after offers* was a problem, since it appeared in a large numbers of the top ranking *segments* (albeit with different lifecycles). However, this was not detected when the analyst was using the standard application. The main reason for this is that the activity appears as four different variant due to its lifecycle attribute. This meant that in the timing overview, the four variants of *W_Call after offers* were all completely average (around 5 days), which did not stand out against the other activities.

Regarding the BPIC 2018 [24] dataset, every participant using our solution noted that, if the value from case attribute *area* was relatively high, the case would be handled quicker (i.e., cases in Class 1 would have, on average, significantly larger values for *area*). This conclusion was not reached by any of the participants using the standard application. The main reason for this discrepancy was the high number of different case attributes in BPIC 2018, making comparison in the standard application was too time consuming, due to analyst not knowing which attributes to combine.

Answering question 5: After the participant was done with both tests, they where asked how they experienced using our *segment analysis*, and our visualization, compared to the standard AppOne application. The following paragraphs denote their answers.

All of the participants noted of the following statement (albeit not in exactly the same words): There is currently (using standard AppOne) no easy way of comparing cases with different throughput times in the same segments. Additionally, seeing the effect of these differences, with respect to other metrics, such as case attributes, event attributes and process flows is almost impossible in the standard AppOne application. According to the participants, currently, no dashboard exist that have this functionality. The dashboards need to made custom per client, which is very time consuming and inconvenient. They noted that our approach and visualization is able to add this exact functionally to contemporary process mining tools. One participant told us: "Even if the standard application would be able to make the same type of comparisons, the analysts would have no way of knowing *what* to compare. Your solution does make it possible for an analyst to know what to compare, since it pre-processes data based on performance."

Both participant 1 and 3 noted that the big advantage of our approach and visualization, compared to contemporary solutions, is that our approach provides a starting point from which to analyze your data. Contemporary solutions just show you which transition is slow (e.g. by making an edge red, or thick), but there is no further analysis than that. Our visualization not only displays the potential bottleneck, it also displays the distribution of performance in that transition, and provides tools to analyze this distribution further. The same participants also noted that, if an analyst did not have access to our approach and visualization, it would take an analyst multiple days of work to reach the same conclusions.

There is a significant amount of domain knowledge needed to understand processes, but this domain knowledge is not always easily available to the analyst. Our approach pre-processes as much information from the event logs as possible, without having access to this domain knowledge. All analysts agreed that our solution allows them to present list of interesting findings and questions to the client, which significantly reduces the time needed to understand the process and the time to discover initial bottlenecks.

8.2.1 Discussion and conclusions

Based on the evaluations discussed in the previous sections, we acknowledged that identifying and detecting root causes for bottlenecks still require a significant amount of work. However, our solution is able to pre-process data, and provide a set of tools which analysts can use to aid them in analyzing processes.

We evaluated the correctness of our approach by applying it on BPIC datasets, and compared our findings, with those of the BPIC reports. We found that our approach is able to discover the same performance related conclusions as presented in the reports. For most of these conclusions, our approach is able to discover them without the analyst having to manually perform any processing, calculation and aggregation steps. Additionally, the approach proved flexible enough to handle different input datasets, and still report similar conclusions. We also note that for most conclusions, we did not have to go further than our first board, meaning that there is a potential to expand on the conclusion even further.

We evaluated the usefulness and understandability of our approach by performing several validation tests with ProcessGold analysts. The participants acknowledged that the approach and corresponding visualization were understandable. Additionally, every participants noted that our approach added new and useful functionally to their existing process mining analysis toolset.

We conclude that *segment analysis* supports the user in identifying and analysing bottlenecks. By observing the proposed dashboards of Section 7.1 to 7.5, and the conclusions of our evaluation in Chapter 8, we demonstrated that we solved **RQ 3:** *Can the outcome of the performance class analysis be aggregated and visualized in such a way, as to support bottleneck identification?*

Chapter 9

Conclusions

In the final chapter of this thesis, we elaborate on the results of this research. In Section 9.1, we discuss the development and results of our solution, with respect to research questions discussed in Chapter 3. Finally, in Section 9.2, we discuss limitations of our approach, and present several possibilities for future work.

9.1 Guided bottleneck identification

In the introduction, we discussed the importance of the *time perspective* of process mining. Furthermore, we discussed the research done by Fahland et al. [11], in which the authors highlighted that contemporary process mining techniques provide only limited insight into performance data, and that *variability in performance* exists within event logs. Additionally, we discussed that the (automatic) extraction of performance information and analysis of the variability in performance had not been researched thus far. Therefore, this thesis aimed to answer the following research question: *Can variability in performance be automatically discovered, analyzed, and visualized to facilitate bottleneck identification?*

To answer this research question, we divided our research into three subjects: *Variability Discovery*, *Variability Analysis*, and *Variability Visualization*. To address *Variability Discovery*, we developed an algorithm which extracts segment information from event logs, according to the methods described by Fahland et al. [11]. Additionally, we proposed several techniques to enhance common, but nonstandard event log variants (i.e., event logs with *start times*, or *lifecycle information*). Using research done by Fahland et al. [11], we confirmed that *variability in performance* is present within event log *segments*. These *segments* consist of time intervals, which have multimodal distributions. To separate these multimodal distributions, we reviewed the literature to discover an appropriate technique for classifying multimodal performance data. We found *Jenks Natural Breaks* (Jenks) to be a fitting classification technique. This technique separates performance data by minimizing the *variance* within each class. The number of classes Jenks generates is a parameter. We developed an approach for automatically setting this parameter, to approximating an optimal number of classes, in which to classify performance data. Finally, we applied the algorithm on a Business Process Intelligence Challenge 2017 dataset, which showed that the technique was able to separate the multimodal performance data into distinct performance classes, which resulted in each segment being separated into a minimum of two groups, and a maximum of seven groups, which, according to our research, is sufficient.

To address *Variability Analysis*, we extended the *segment* performance classes by including *overall* performance classes. We generate the *overall* performance classes, by calculating, for each class in the segment performance classes, the throughput time of each case in the class, and aggregate this information. Additionally, to judge whether a segment contains a potential bottleneck, and

to be able to automatically analyze performance classes, we formulated several analysis questions, and proposed several measures, which each characterize a different part of the performance classes (i.e., size, overlap, throughput etc.). To answer these questions, we developed three algorithmic solutions, named *effect size*, *lost time*, and *total weighted impact*, which automatically answer these questions, called *measures*. These *measures* were developed either, by using techniques from literature, or developed custom. Finally, since we cannot compare *measures* directly to each other, we developed a ranking which compares segments *relative* to other segment, which ranks the segments based on likelihood of containing bottlenecks, from highest to lowest.

To address *Variability Visualization*, we set out to visualize the outcome of the *Variability Analysis* in the ProcessGold platform. To support bottleneck identification, we analyzed process mining literature, and found that process mining analysis consists of four perspectives, namely *time, case, organization*, and *control-flow*. To capture the *time* perspective, we created a dashboard which shows the performance characteristics of each segment. To capture the *case* perspective, we implemented *case attribute frequency analysis* research by Verhoef [15]. To capture the *organizational* perspective, we developed a dashboard to analyze event attribute frequency between each performance class. To capture the *control-flow* perspective, we developed a dashboard to analyze process-flows between performance classes. The combination of these dashboard gives the analyst a complete set of tools to analyze the performance classes, and assist in root-cause bottleneck analysis.

Finally, to validate our approach, we evaluated the understandability, usefulness, and correctness of our *segment analysis*. The correctness was evaluated by comparing the outcome of our segment analysis to the findings discovered by the winners of the Business Process Intelligence Challenge 2017. We found that most performance related findings were able to be recreated using our approach. Furthermore, we noted that most of these findings could be recreated using only the starting dashboard of our visualization. The understandability and usefulness of our approach was evaluated by conducting validation tests with ProcessGold analysts. Participants were unanimously positive on the functionality our *segment analysis* added to their process mining tool set.

9.2 Limitations and future work

One limitation of our approach is the presence of user parameters, such as *RGVFC_threshold* (discussed in Section 5.4.2) and *Minimum segment frequency* (discussed in Section 8.1.1). Setting these parameters correctly is a non-trivial task, and may require the user several tries before getting a satisfactory result. For the *RGVFC_threshold*, we gave a range of safe options to choose from in Section 5.4.2. However, for *minimal segment frequency*, we currently cannot give a safe range for, since it is very dependant on the characteristics of the event log. Future work could include creating an approach to automate fine-tuning of these parameters.

Another limitation of our approach is the fact that, if a segment occurs more than once, only the first occurrences of this segment is counted in our analysis. We made this decision because we wanted all our observations to be *independent* of one another. For example, if a segment occurred many times within a case, the *overall* performance class would hold multiple instances of the same case, which would skew the averages of the *overall* class towards that case. Therefore, we recommend to keep in mind that analyzing event logs where a high percentage of transitions are loops (i.e., transitions from an activity to the same activity) may result in misleading and/or wrong results.

A limitation of our evaluation approach regarding the BPIC reports, is that we lack the necessary domain knowledge to fully understand the processes in BPIC 2017. This means that we were only able to verify whether a bottleneck that was found by the analysts of the BPIC reports, could also

be found using our approach. We were not able to definitively conclude that we identified more bottlenecks, or than we identified bottlenecks quicker, compared to the BPIC reports. While we do see many signs that our approach does assist analysts in identifying bottlenecks, this limitation makes it currently impossible to definitively conclude that our approach is an improvement on standard analysis techniques.

Another limitation (which is also a candidate for future work), is that the *Event Attribute* visualization does not contain on option to check the *hand over* of event attributes. *Hand over* denotes the difference in event attribute values/frequencies, when a case goes from one activity, to the next. This could be a very powerful tool which fits perfectly with our *segment* based analysis. We we not able to implement this functionality successfully in our visualization (mainly due to time limitations, not technical ones).

Additionally, we recommend a possible future extension which is useful for future practical application. Recall the performance class discussed in Section 6.2. In this section we explained how we extended the *segment* performance classes, by calculating their *overall* performance classes. During this research, we calculated two additional performance classes, namely, *elapsed time* performance classes, and *remaining time* performance classes. For the *elapsed time* performance classes, we calculate for every class in the *segment*, the time between the start time of the case, and the beginning of the segment. For the *remaining time* performance classes, we calculate for every class in the *segment*, the time between the end time of the segment, and the end of the case. Figure 9.1 shows an extended version of Figure 6.3, which shows where in the process, the performance classes are located. Each of the new performance classes have their own set of characteristics and correlations. To give an example, during our research, we noted that one *segment* had *elapsed time* performance classes which where decreasing. This meant that whether *more* time was spend on the activities before the segment occurred, the *less* time was spent in the segment itself. These additional performance classes hold a wealth of analysis opportunities. Possible future work could include creating a set of performance class measures, just like we did for the *segment* and *overall* classes.

Finally, the algorithms are currently implemented in Python, with the visualizations being separately implemented in the ProcessGold platform. Future work could include combining both into a plugin for the widely used open-source process mining software ProM. [40].

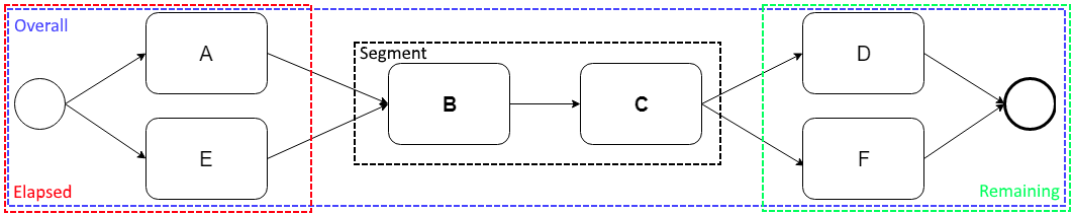

Figure 9.1: *Segment* and *Overall* processes, extended with *Elapsed time* performance classes and *Remaining time* performance classes

Bibliography

[1] Mathias Weske. Business process management architectures. In *Business Process Management*, pages 333–371. Springer, 2012. 1

[2] Wil van der Aalst, Marcello La Rosa, and Flávia Maria Santoro. Business process management. *Business Information Systems Engineering*, 58:1–6, 2008. 1, 2

[3] Wil 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. 1

[4] Laura Mruter and Nick R.T.P. Van Beest. Redesigning business processes: A methodology based on simulation and process mining techniques. *Knowledge and Information Systems*, 21(3):267–297, 2009. 1

[5] Wil van der Aalst. *Process Mining*, volume 5. 2016. 2, 6, 7, 14, 25

[6] Wil Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer, 2011. 2, 11, 44, 50, 52

[7] Wil Van Der Aalst. Data science in action. In *Process Mining*, pages 3–23. Springer, 2016. 2, 3

[8] Laura Măruşter and Nick RTP van Beest. Redesigning business processes: a methodology based on simulation and process mining techniques. *Knowledge and Information Systems*, 21(3):267, 2009. 3

[9] B. F. van Dongen, Ronald A Crooy, and Wil van der Aalst. Cycle time prediction: When will this case finally be finished? In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 319–336. Springer, 2008. 3

[10] Wil MP Van der Aalst, M Helen Schonenberg, and Minseok Song. Time prediction based on process mining. *Information systems*, 36(2):450–475, 2011. 3

[11] Vadim Denisov B, Dirk Fahland, and Wil van der Aalst. Unbiased , Fine-Grained Description of Processes Performance from Event Data. 1:139–157, 2018. 3, 9, 10, 14, 15, 16, 64

[12] Sander J.J. Leemans, Dirk Fahland, and Wil van der Aalst. Using life cycle information in process discovery. *Lecture Notes in Business Information Processing*, 256(i):204–217, 2016. 7

[13] Wil van der Aalst and Schahram Dustdar. Process mining put into context. *IEEE Internet Computing*, 16(1):82–86, 2012. 7

[14] 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, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011. 7

[15] Céline Verhoef. Analyzing influence of case properties on business processes. Internship report, Eindhoven University of Technology, 2018. 11, 22, 48, 65

[16] International conference on business process management. 18, 55

[17] H.M. Walker and J. Lev. *Elementary statistical methods*. International series in decision processes. Holt, Rinehart and Winston, 1969. 20, 27

[18] Bin Jiang. Head/Tail Breaks: A New Classification Scheme for Data with a Heavy-Tailed Distribution. *Professional Geographer*, 65(3):482–494, 2013. 20, 27

[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press. 20, 27

[20] GF Jenks. Optimal data classification for choropleth maps occasional paper no 2. *University of Kansas, Department of Geography*, 1977. 20, 27, 29

[21] Toshihiro Osaragi. Classification methods for spatial data representation. *Osaragi, Toshihiro (2002) Classification methods for spatial data representation. Working paper. CASA Working Papers (40). Centre for Advanced Spatial Analysis (UCL), London, UK.*, 07 2008. 20, 28

[22] B.F. Van Dongen. Bpi challenge 2017. 20, 22, 23, 25, 32, 33, 55, 61

[23] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 2013. 21, 37, 39

[24] B.F. Van Dongen F. Borchert. Bpi challenge 2018. 22, 23, 32, 61

[25] W. Steeman. Bpi challenge 2012. 23, 25, 32

[26] B.F. Van Dongen. Bpi challenge 2014. 23, 32

[27] B.F. Van Dongen. Bpi challenge 2015. 23, 32

[28] B.F. Van Dongen. Bpi challenge 2019. 23, 32

[29] M.A.W. de Roode. Inter-case resource dependencies. 2018. 25

[30] Norman Cliff. QUANTITATIVE METHODS IN PSYCHOLOGY Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions. *Psychol Bull*, 114(3):494–509, 1993. 26

[31] William H Kruskal, W Allen Wallis, William H Kruskal, and W Allen Wallis. Use of Ranks in One-Criterion Variance Analysis Stable URL : http://www.jstor.org/stable/2280779 JOURNAL OF THE AMERICAN ANALYSIS. 47(260):583–621, 2016. 26

[32] Douglas G. Bonett and Thomas A. Wright. Sample size requirements for estimating Pearson, Kendall and Spearman correlations. *Psychometrika*, 65(1):23–28, 2000. 26

[33] Yue Lin. A comparison study on natural and head/tail breaks involving digital elevation models, 2013. 28

[34] Walter D Fisher. On grouping for maximum homogeneity. *Journal of the American statistical Association*, 53(284):789–798, 1958. 29

[35] George A. Miller. The Magical Number Seven, Plus or Minus Two Some Limits on Our Capacity for Processing Information. *Psychological Review*, 101(2):343–352, 1955. 32

[36] N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, 2001. 32

[37] Shlomo S. Sawilowsky. New Effect Size Rules of Thumb. *Journal of Modern Applied Statistical Methods*, 8(2):597–599, 2009. 39

[38] Julie Robbrecht Liese Blevi and Lucie Delporte. Process mining on the loan application process of a dutch financial institute. 2017. 56, 57, 58, 59

[39] Ariane Rodrigues, Cassio Almeida, Daniel Saraiva, Felipe Moreira, Georges Spyrides, Guilherme Varela, Gustavo Krieger, Igor Peres, Leila Dantas, Mauricio Lana, Odair Alves, Rafael Frana, Ricardo Neira, Sonia Gonzalez, William Fernandes, Simone Barbosa, Marcus Poggi, and Hélio Lopes. Stairway to value: Mining the loan application process. 56, 58

[40] B F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP Van Der Aalst. The prom framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*, pages 444–454. Springer, 2005. 66

Appendix A

RGVFC_Threshold experiments

A.1 0.1% tests

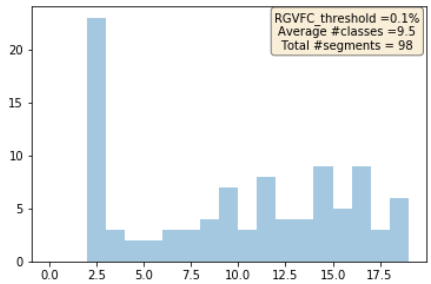

Figure A.1: *0.1% tests, BPIC 2012*

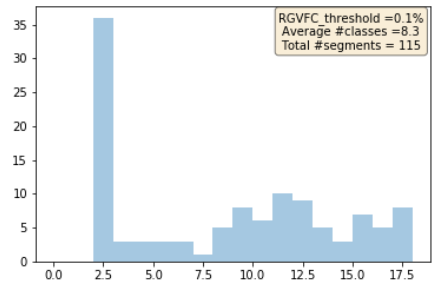

Figure A.2: *0.1% tests, BPIC 2017*

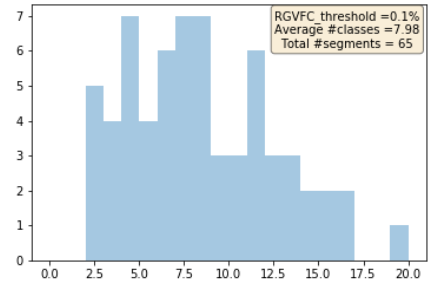

Figure A.3: *0.1% tests, BPIC 2018*

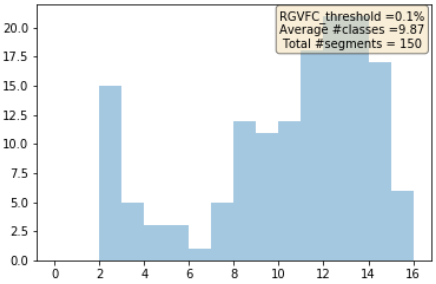

Figure A.4: *0.1% tests, BPIC 2019*

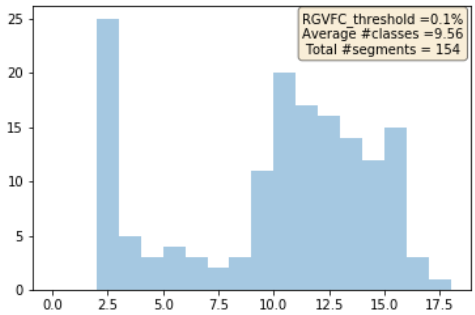

Figure A.5: *0.1% tests, PG Internal*

A.2 0.5% tests

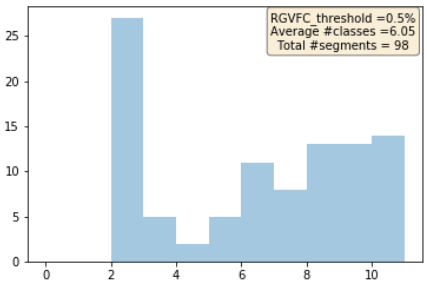

Figure A.6: *0.5% tests, BPIC 2012*

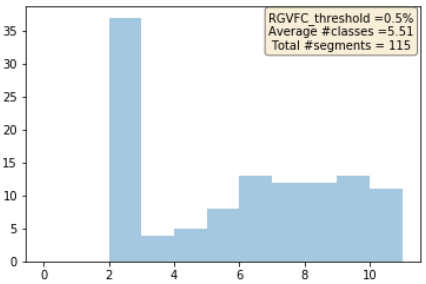

Figure A.7: *0.5% tests, BPIC 2017*

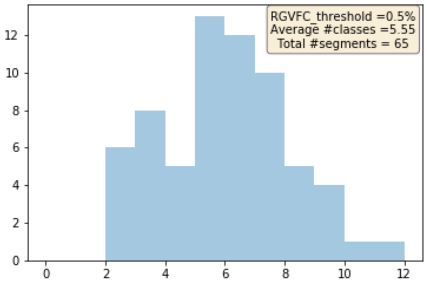

Figure A.8: *0.5% tests, BPIC 2018*

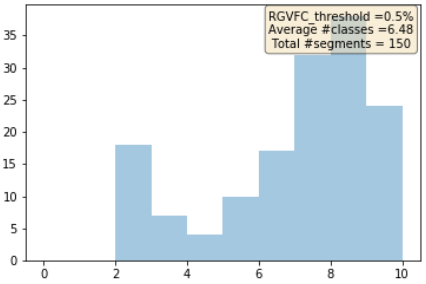

Figure A.9: *0.5% tests, BPIC 2019*

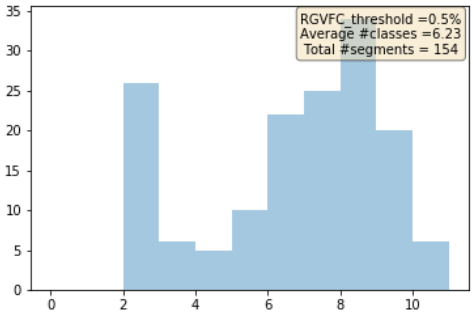

Figure A.10: *0.5% tests, PG Internal*

A.3 1.0% tests

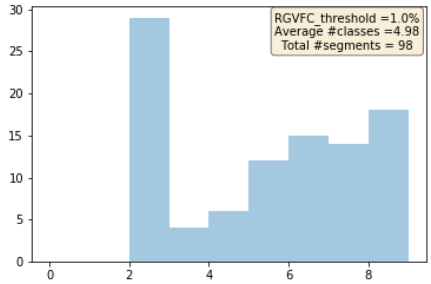

Figure A.11: *1.0% tests, BPIC 2012*

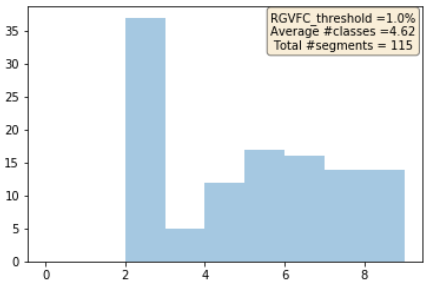

Figure A.12: *1.0% tests, BPIC 2017*

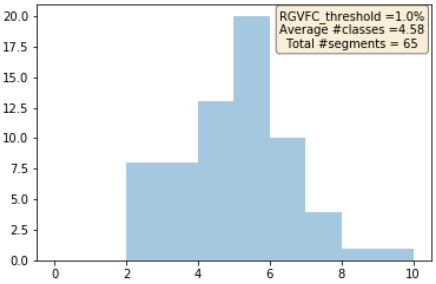

Figure A.13: *1.0% tests, BPIC 2018*

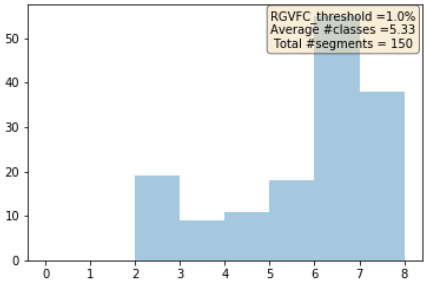

Figure A.14: *1.0% tests, BPIC 2019*

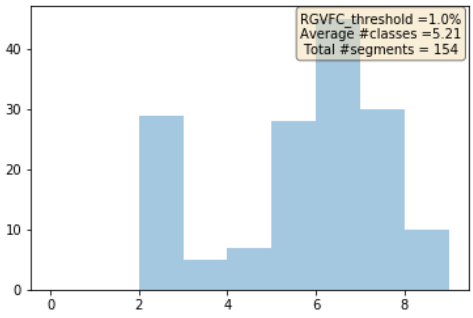

Figure A.15: *1.0% tests, PG Internal*

A.4 2.5% tests

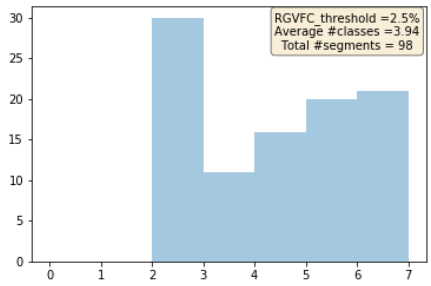

Figure A.16: *2.5% tests, BPIC 2012*

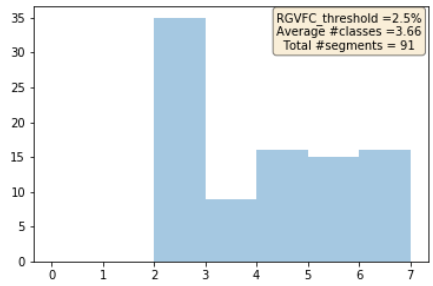

Figure A.17: *2.5% tests, BPIC 2017*

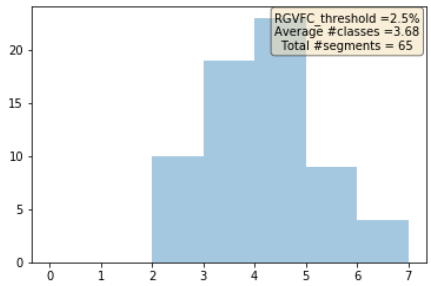

Figure A.18: *2.5% tests, BPIC 2018*

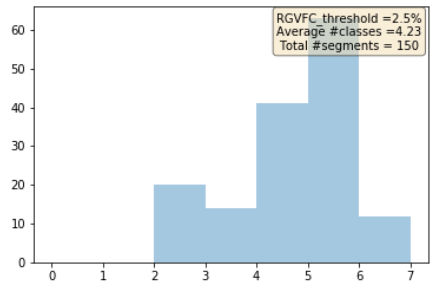

Figure A.19: *2.5% tests, BPIC 2019*

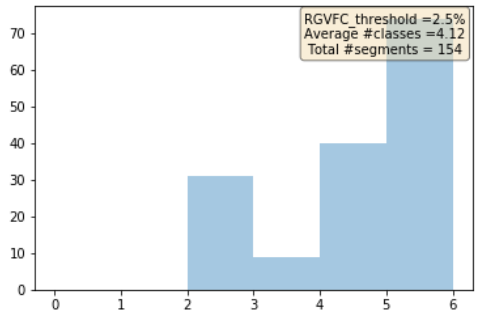

Figure A.20: *2.5% tests, PG Internal*

A.5 5.0% tests

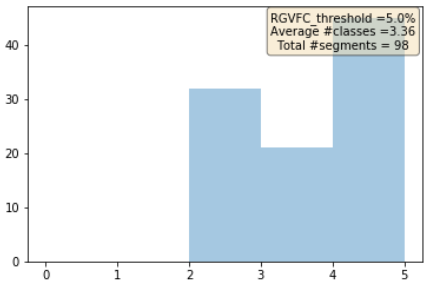

Figure A.21: *5.0% tests, BPIC 2012*

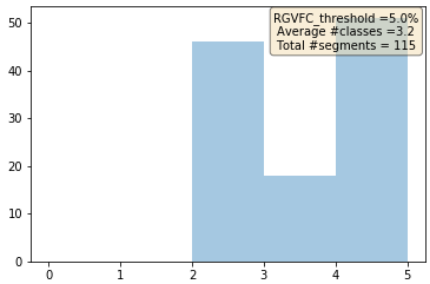

Figure A.22: *5.0% tests, BPIC 2017*

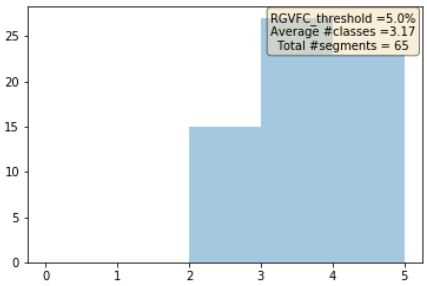

Figure A.23: *5.0% tests, BPIC 2018*

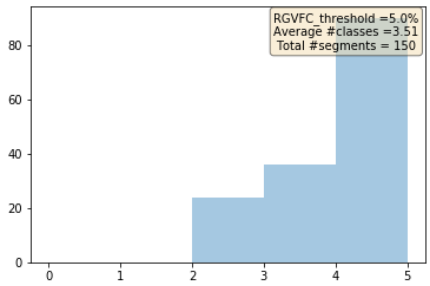

Figure A.24: *5.0% tests, BPIC 2019*

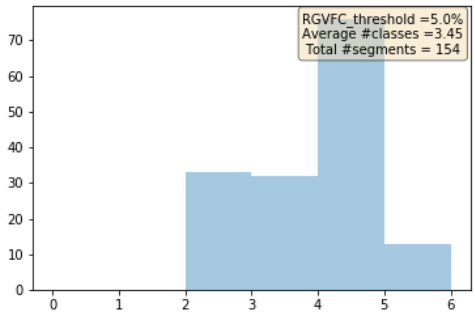

Figure A.25: *5.0% tests, PG Internal*

A.6 10.0% tests

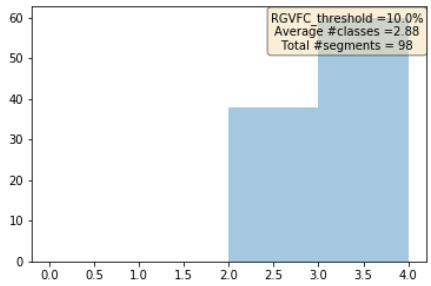

Figure A.26: *10.0% tests, BPIC 2012*

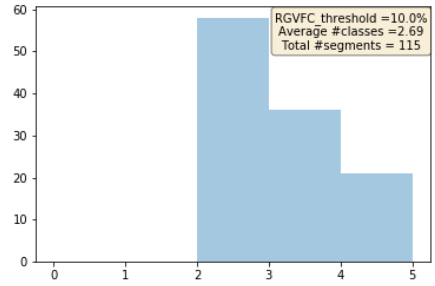

Figure A.27: *10.0% tests, BPIC 2017*

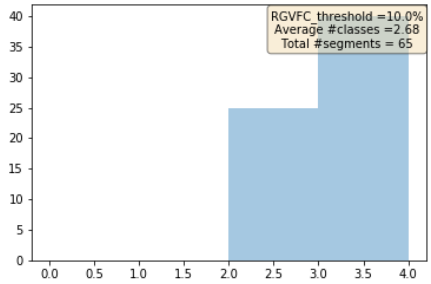

Figure A.28: *10.0% tests, BPIC 2018*

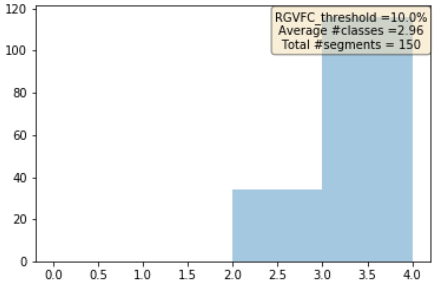

Figure A.29: *10.0% tests, BPIC 2019*

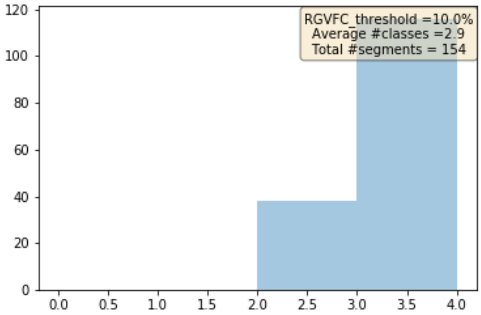

Figure A.30: *10.0% tests, PG Internal*

A.7 25.0% tests

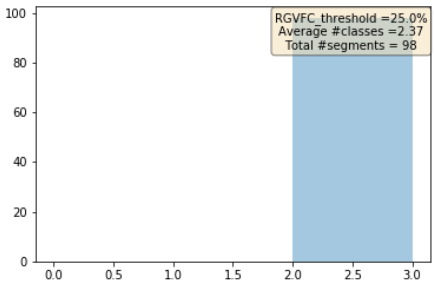

Figure A.31: *25.0% tests, BPIC 2012*

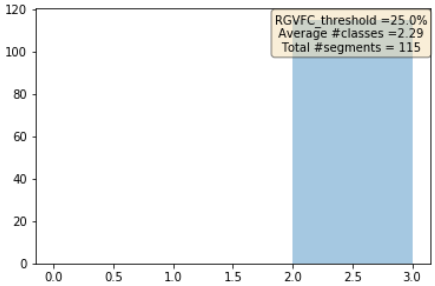

Figure A.32: *25.0% tests, BPIC 2017*

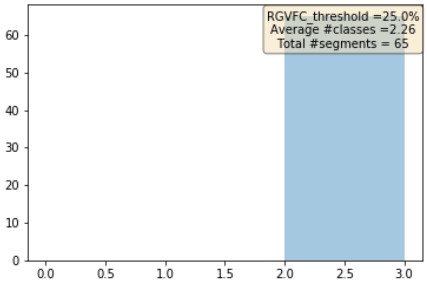

Figure A.33: *25.0% tests, BPIC 2018*

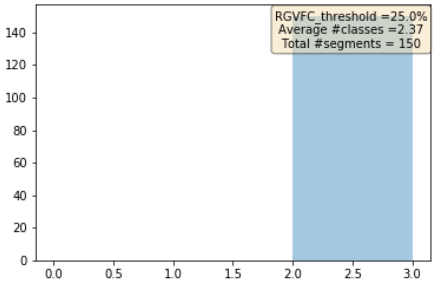

Figure A.34: *25.0% tests, BPIC 2019*

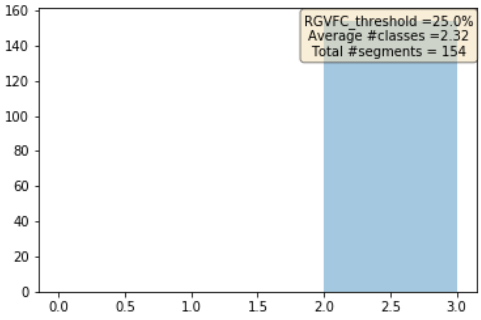

Figure A.35: *25.0% tests, PG Internal*

Appendix B

BPI Challenge 2017

▼ Segment name	#Cases	Classes	Effect size ranking	Lost time ranking	Weighed impact ...	Total
W_Call after offers+suspend –> A_Cancelled+complete	7.573	2	48	48	48	144
W_Call after offers+suspend –> W_Call after offers+ate_abort	21.033	3	42	46	46	134
W_Call after offers+schedule –> W_Call after offers+withdraw	2.010	3	47	36	47	130
W_Call incomplete files+suspend –> O_Accepted+complete	4.743	3	41	45	43	129
W_Call after offers+suspend –> O_Create Offer+complete	3.489	3	45	42	42	129
O_Sent (mail and online)+complete –> W_Call after offers+ate_abort	1.384	4	44	38	45	127
W_Call after offers+suspend –> W_Call after offers+resume	24.009	4	35	47	44	126
O_Sent (mail and online)+complete –> W_Call after offers+resume	1.255	4	46	33	41	120
W_Call incomplete files+suspend –> W_Call incomplete files+ate_abort	10.066	4	40	40	38	118
W_Validate application+suspend –> W_Validate application+ate_abort	9.578	3	32	41	37	110
W_Complete application+suspend –> A_Accepted+complete	9.457	4	43	35	32	110
W_Complete application+suspend –> W_Complete application+resume	11.760	4	36	39	34	109
W_Validate application+suspend –> W_Validate application+resume	11.487	3	22	44	39	105
A_Concept+complete –> W_Complete application+start	15.065	4	24	43	35	102
W_Call incomplete files+suspend –> O_Create Offer+complete	1.035	4	38	30	33	101
W_Validate application+suspend –> O_Accepted+complete	8.674	3	27	37	36	100
W_Call incomplete files+suspend –> W_Call incomplete files+resume	14.500	5	37	31	30	98
A_Concept+complete –> A_Accepted+complete	9.231	4	34	32	31	97
W_Validate application+suspend –> A_Denied+complete	1.725	3	15	34	40	89
A_Accepted+complete –> O_Create Offer+complete	29.966	5	33	28	24	85
O_Created+complete –> O_Sent (mail and online)+complete	30.386	6	39	25	10	74
W_Complete application+start –> A_Accepted+complete	6.749	5	26	22	21	69
W_Call incomplete files+resume –> W_Call incomplete files+suspend	13.684	6	29	19	16	64
W_Handle leads+suspend –> W_Handle leads+resume	780	5	14	21	28	63
W_Validate application+resume –> O_Accepted+complete	3.019	4	19	17	25	61
W_Handle leads+schedule –> W_Handle leads+start	3.643	3	2	29	29	60
A_Complete+complete –> W_Call after offers+suspend	30.204	6	10	27	23	60
W_Validate application+resume –> W_Validate application+suspend	9.768	6	12	24	20	56
W_Call incomplete files+resume –> W_Call incomplete files+complete	1.777	3	30	8	18	56
O_Returned+complete –> W_Validate application+suspend	18.507	5	13	23	19	55
W_Validate application+resume –> W_Validate application+complete	5.042	5	1	26	26	53
W_Validate application+resume –> A_Denied+complete	1.496	4	11	14	27	52
A_Validating+complete –> W_Validate application+suspend	11.416	5	20	18	14	52
A_Incomplete+complete –> W_Call incomplete files+suspend	14.673	5	17	20	13	50
A_Accepted+complete –> W_Complete application+suspend	1.322	4	28	7	15	50
W_Complete application+resume –> A_Accepted+complete	6.043	4	9	15	22	46
W_Handle leads+schedule –> W_Handle leads+withdraw	16.802	6	31	4	8	43
W_Complete application+resume –> W_Complete application+suspend	8.098	7	16	12	11	39
A_Concept+complete –> W_Complete application+suspend	3.806	6	25	9	5	39
W_Call after offers+resume –> W_Call after offers+suspend	25.063	5	23	11	4	38
A_Validating+complete –> O_Returned+complete	20.406	4	18	13	7	38
O_Returned+complete –> W_Validate application+complete	2.065	5	8	10	17	35
W_Complete application+start –> W_Complete application+suspend	11.819	6	5	16	12	33
A_Validating+complete –> W_Validate application+complete	1.846	4	21	5	6	32
W_Handle leads+start –> W_Handle leads+complete	2.743	4	4	6	9	19
W_Validate application+start –> A_Validating+complete	21.870	4	6	2	2	10
O_Create Offer+complete –> O_Created+complete	31.509	3	3	3	3	9
A_Create Application+complete –> A_Submitted+complete	20.423	3	7	1	1	9

Figure B.1: Ranked segments, BPIC 2017 *with* lifecycle information

▼ Segment name	#Cases	Classes	Effect size ranking	Lost time ranking	Weighed impact ranking	Total
W_Call after offers –> A_Cancelled	8.539	2	26	26	26	78
W_Call incomplete files –> O_Accepted	4.783	3	23	25	25	73
W_Call after offers –> O_Create Offer	3.756	3	25	23	23	71
O_Sent (mail and online) –> W_Call after offers	3.272	4	22	22	24	68
W_Complete application –> A_Accepted	22.249	4	24	20	16	60
W_Complete application –> O_Create Offer	1.574	3	20	16	19	55
A_Concept –> W_Complete application	22.264	4	11	24	20	55
W_Validate application –> O_Accepted	11.738	3	12	21	21	54
W_Call incomplete files –> O_Create Offer	1.670	4	21	15	18	54
A_Concept –> A_Accepted	9.231	4	17	18	17	52
W_Validate application –> A_Denied	3.254	3	9	19	22	50
A_Accepted –> W_Complete application	1.538	5	18	13	15	46
O_Returned –> W_Validate application	20.675	5	15	14	12	41
O_Returned –> W_Call incomplete files	1.479	4	16	10	14	40
W_Handle leads –> W_Handle leads	20.423	3	7	17	13	37
A_Accepted –> O_Create Offer	29.966	5	14	12	10	36
O_Created –> O_Sent (mail and online)	30.386	6	19	7	5	31
W_Complete application –> W_Complete application	24.649	6	13	9	8	30
A_Complete –> W_Call after offers	30.396	6	5	11	9	25
W_Call incomplete files –> O_Cancelled	1.529	4	1	8	11	20
A_Incomplete –> W_Call incomplete files	14.731	5	8	6	6	20
A_Validating –> W_Validate application	11.662	5	6	5	7	18
A_Validating –> O_Returned	20.406	4	10	4	4	18
O_Create Offer –> O_Created	31.509	3	2	3	3	8
W_Validate application –> A_Validating	21.870	4	3	2	2	7
A_Create Application –> A_Submitted	20.423	3	4	1	1	6

Figure B.2: Ranked segments, BPIC 2017 *without* lifecycle information