LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME
UND DATA MINING

Master Thesis
in Computer Science

Inferring Process Performance Models from Interval Events using the Performance Skyline

Andrea Maldonado

Supervisor:	Prof. Thomas Seidl
Advisor:	Janina Sontheim
Submission Deadline:	30.03.2020

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
This paper was not previously presented to another examination board and has not been published.

Munich, Germany, 30.03.2020

.....
Andrea Maldonado

Abstract

Performance mining from event logs is a central element to manage and improve business processes. Established performance analysis techniques are either based on control-flow models, which simulate all possible execution paths for a process at once, or methods that propose extracting performance features from only one timestamp. This thesis integrates interval-based methods from sequence pattern mining into process mining to discover performance process models from event logs that include both, start and end event timestamps. In addition it introduces the *performance skyline*, which describes the series of events that lead to the worst case duration of a process and enables novel extensions for further process discovery, conformance checking and process enhancement. Executed experiments on real event logs show that presented models assist detecting and classifying trace anomalies into multiple categories.

Contents

Chapter 1

Introduction

1.1 Motivation

Most processes in real life businesses today are complex, constantly changing and frequently not explicitly described [19]. Business processes often remind of black boxes, where input and output are known but not what is inside. Furthermore sometimes a process execution time is not deterministic, and in worst case scenario, often much longer that expected.[7] In today's fast paced industry, businesses may reach their potential by speeding up their processes, and rapidly identifying as well as preventing duplication and mistakes in them. In order to achieve these improvements, it is necessary to understand the process itself. A way to gain knowledge about processes is to detect deviation between their description, which sketches corresponding real executions, and prescription, which details its corresponding expectation [3]. In addition, business processes and information systems require alignment, which in a constantly changing set up also demands continuous attention [19]. Thus methodically and automatically discovering processes is the first step towards process enhancement.

The process mining algorithm has to choose the correct "granularity" for generating the process model, but it has to take advantage of all the available information. For example, in [5], any activity spans over time intervals so, the mining algorithm can exploit this fact in order to extract a better more precise model. A similar problem occurs when data referring to several perspectives is available, as in the case of process performance modeling: it would be desirable to embed all of them into the same representation [6, 4].

1.2 Problem Statement & Scope

Consider the following example: Two people go shopping for groceries together. A typical course for this could be driving to the supermarket, parking the vehicle, returning recyclable bottles, collecting specific grocery items, walking to the cashier, queuing at the cashier, paying, loading groceries into the vehicle and leave. If all these activities are executed sequentially, they all lay in the critical path, which is the aggregation of activities that directly contribute to the overall duration of the process. To shorten the critical path, the shoppers decide to split up. In order to know what activities to run in parallel to make the process more efficient, knowledge about the duration of each event and potential waiting times is necessary.

Although current process mining and performance analysis tools are able to extract knowledge from event logs, as well as to create models that describe the process underneath and some aspects of its performance, these models make only use of single timestamp events. Some real logs offer events with multiple timestamps from where more knowledge about underlying processes can be extracted.

Thus the research question underlying this thesis is: How to extract knowledge and model the performance of processes containing interval events? Process Performance Models resulting from this thesis aim to provide more accurate process discovery as well as enable more extensive conformance checking and enhancements for processes containing interval events. For example, by offering general insights into the performance of a process or by detecting anomalies.

Since this master's thesis aims to provide performance models of interval events that offer insights into real processes too, it will provide experiments on real logs from the data processes of a German online feedback analysis company, TrustYou GmbH. For this purpose these process logs of this company will be investigated regarding anomaly detection using the proposed approach.

1.3 Methodology

This thesis's challenge was divided into multiple parts. For better understanding, current approaches in the research areas of Process, Performance and Sequence Pattern Mining were researched. Different available practices were found, examined and combined into a novel method. This method can be divided into the following steps: Log extraction and preprocessing; process

and performance model notation; and visualization. After conceptualizing and implementing this new solution, it was evaluated for the task of anomaly detection using real trace logs.

1.4 Structure

In the next chapter, background knowledge and related work, all necessary concepts to understand the context of this thesis will be provided. This includes definitions and notations that will be used throughout the rest of this thesis for interval events, process mining. Their approaches and limitations to handle temporal data from processes will also be described in this in this chapter. Additionally this chapter will also cover information on interval-based sequence pattern mining concepts, like allen's interval algebra, and interval's geometric representation.

Combining mentioned areas of research in chapter 3, will produce the basis for a novel approach for performance models for interval events. This chapter will present the central contribution of this thesis, which is the conception and visualization of the performance skyline and other trace set models, which increase the knowledge that can be extracted from current approaches. In this chapter presented concepts and definitions for the task of trace anomaly detection will be provided to test the novel methods.

Computational experiments and discussion, chapter 4, will examine the presented concept's suitability for the task of anomaly detection on the real data set also comparing this solution to current existing similar systems in the science community, discussed in chapter 2. Finally chapter 5, conclusion, will close presenting achievements of this thesis and further expansion possibilities as future work on this topic.

Chapter 2

Background Knowledge and Related Work

Process performance models for interval events being this work's focus, this chapter offers definitions and notation, which will be used throughout the rest of this thesis. In the following section, a brief introduction into process mining is provided. Next, section 2.2 shows methods that mine processes focusing on the performance aspect. This chapter will also present available state of the art methods from sequence pattern mining for modeling temporal intervals in section 2.3 and section 2.4.

2.1 Process Mining

Process mining builds on two pillars: Data mining, and process modeling. Data mining can be defined as the analysis of, often large, data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner. A process model's purpose is to give a representation of for example a business process using information about every execution of it.[19] Processes data contains a series of performed actions, called activities. The finite set of all valid activities, depends on the domain of the process and is referred to as A. A single execution of an activity is called an event. Even though timestamps might contain date and time in a specific format, most importantly they can be ordered. For simplicity it is assumed that case ids and timestamps can be identified using natural numbers.

Definition 2.1.1 *An **event** $e = (c, a, t) \epsilon \mathbb{N} \times A \times \mathbb{N}$ is as an tuple consisting of case id c, an activity id, a a timestamp t.*

Furthermore formally the data set containing information about the executions is called a log.

Definition 2.1.2 *A **log** $L : \mathbb{N} \times A \times \mathbb{N} \to \mathbb{N} \cup 0$ is a multiset of events.*

A log can contain multiple traces, describing single executions of the process.

Definition 2.1.3 *Let L be a log and c a case id, a **trace** is a finite sequence of events $\sigma_c = (e_1, e_2, \ldots, e_n))$, where $\pi_c(e_i) = c$ for all $1 \leq i \leq n$. Additionally, the trace is ordered based on the event timestamps, formally $\pi_t(e_i)) \leq \pi_t(e_j)) \Leftrightarrow i \leq j$.*

A trace can contain multiple events of a single activity, which means an activity can happen many times. If two events have an equal timestamp and activity id, they are consider the same event. Although a trace compounds ordered activities, it does not necessarily describes a strict linear structure. Events can have a different activity id and equal timestamp, which means they are two different events which happen simultaneously.

For most techniques every event in a log needs at least a case id, an activity id and a ordinal timestamp to extract knowledge about the corresponding process.[19] Consider the example log in figure 2.1 with case ids `1` and `4`; activities `register request`, and `check ticket` and multiple ordered timestamps.

Using the log and a process model, such as the alpha-miner introduced in [19], results in figure 2.2, which describes the full behavior by deriving the "100% model" of the process, covering all cases observed in the log. figure 2.2's example portrays the process as follows:

All traces of this process start with the activity `register request`, then they either `examine thoroughly` or `check ticket` next. The following activity for both is `decide`, then finally all traces in this process end with `reject request`.

Although the process structure can be explained by models using solely case id, activity id and timestamp, enriching logs with

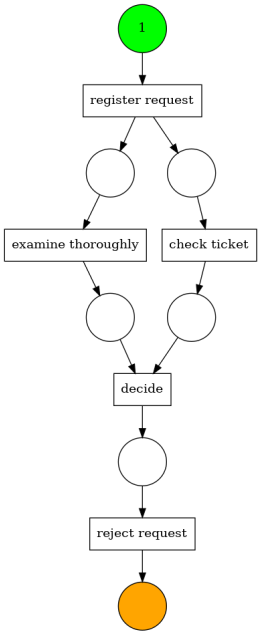
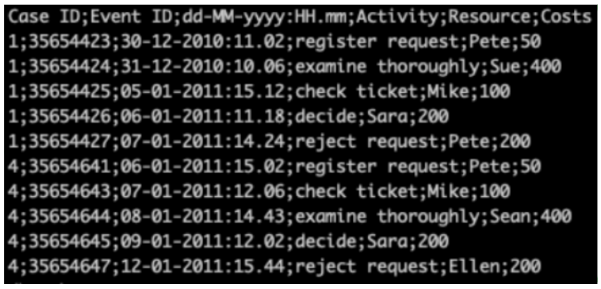


Figure 2.2: Process mining techniques build process models from event logs.



a) Log example

Case id	Activity id	Timestamp
1	`register request`	30-12-2010:11.02
1	`examine thoroughly`	31-12-2010:10.06
1	`check ticket`	05-01-2011:15.12
1	`decide`	06-01-2011:11.18
1	`reject request`	07-01-2011:14.24
⋮	⋮	⋮

b) Extracted events from *a)* with case ids and activity ids

Figure 2.1: Example of log extraction case ids and activity ids

a time perspective may teach about the timing and frequency of events in a process. When events regard timestamps beyond ordering, it is possible to discover bottlenecks, measure service levels, monitor the utilization of resources, and predict the remaining processing time of running cases [7].

2.2 Performance Mining

Performance mining considers the temporal aspect of processes to understand them better and increase their efficiency. The performance spectrum miner [7] maps all observed flows between activities together regarding their performance over time. It uses an horizontal axis, representing time and several parallel horizontal lines representing each activity that is part of the process. Every trace is drawn as a line passing through every horizontal at the time a certain event for the corresponding activity starts. For the example in figure 2.3, the trace is with `case 1` is depicted in the following manner:

`Case 1` consists of three activities. It starts at the point of timestamp `t1` on the `activity 1` horizontal. Next, `activity 2` starts at `t2`, thus a line is plotted between both points. The following event, `activity 3`, starts at `t3`, consequently the line reaches `t3`. This procedure is repeated until all events

of a case have been listed. In this example, `activity 3` is the last event.

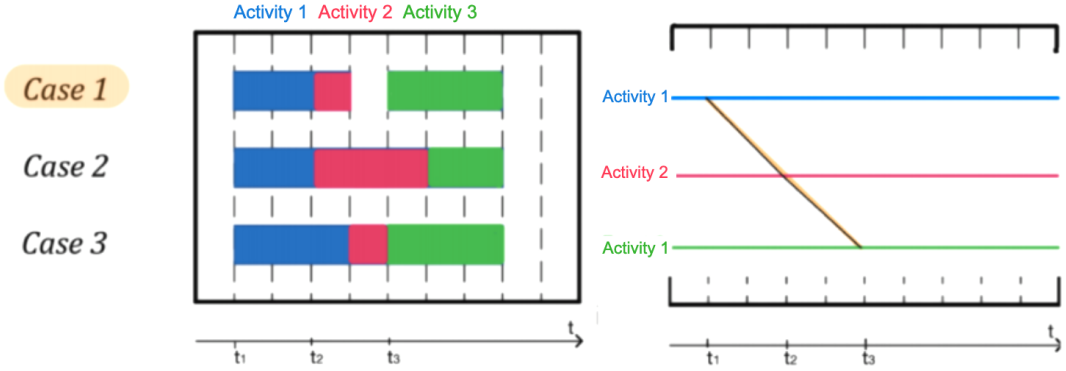


Figure 2.3: On the left a schematic representation of three example cases, and on the right, their corresponding performance spectrum.

The performance spectrum can be also used to distinguish performance patterns as batching behaviour, which describes different traces running the same activity simultaneously [10]. In spite of this, using only one timestamp leads to the assumption that every event ends exactly when the next one starts, forcing models to overlook some aspects like waiting time, actual event duration and actual trace duration i.a. service time. Not extracting this knowledge restricts models an pattern detection methods derived from it.

Other models, as for example PROM's dotted chart [14], use two-dimensional

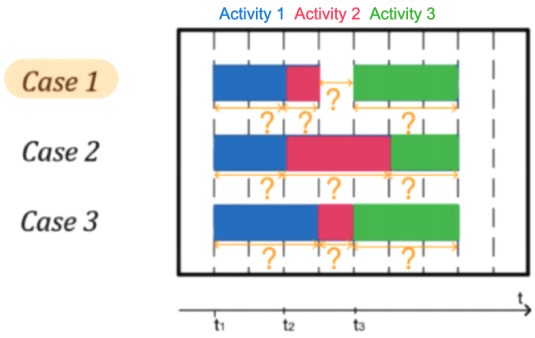


Figure 2.4: Limitations of using only one timestamp [10].

space projections with start time in the horizontal axis and case ids in the vertical axis to describe processes. Nevertheless, as all presented methods so far it is also limited by using a single timestamp. The next section describes a richer knowledge extraction alternative for event logs bearing multiple timestamps instead of a single one.

For this reason section 3.1 introduces *interval events*. Interval events have already been defined by others [13], but slightly differently than in this thesis, since in the lack of a second timestamp in most real logs, it is implicit that an interval event ends exactly when the next one begins [7]. This hinders to identify gaps between two events, for instance on `case 1` between `activity 2` and `activity 3` in figure 2.4.

From another point of view, in sequential pattern mining temporal intervals describe time ranges with a start and an end. Relations between them can be described using Allen's interval algebra, which will be presented next.

2.3 Allen's Interval Algebra

A temporal interval or event A is defined by its start t_A^+ and its end t_A^-, where $t_A^+ < t_A^-$. Let A and B be two temporal intervals, Allen's interval algebra [2] defines thirteen possible chronological relations between them, as shown in figure 2.5. Being `A` and `B` two temporal intervals projected as lines, and their starts and ends be projected as points, the following relationships are defined.

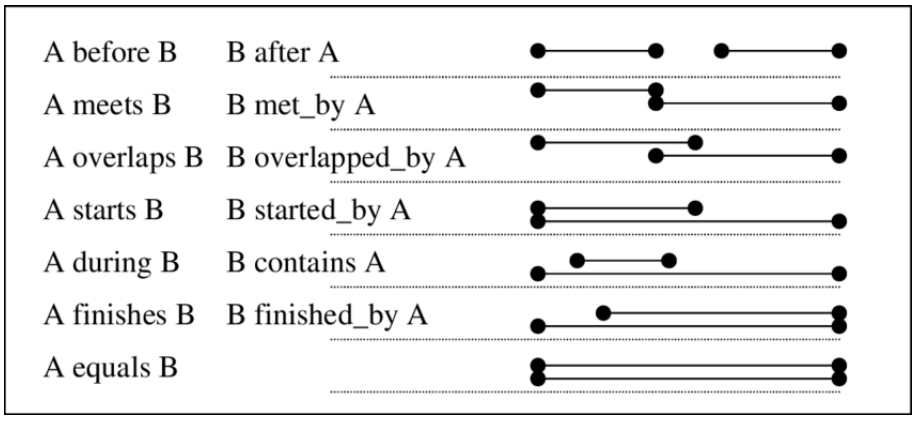


Figure 2.5: thirteen relations capture the possible interactions between two temporal intervals. [18].

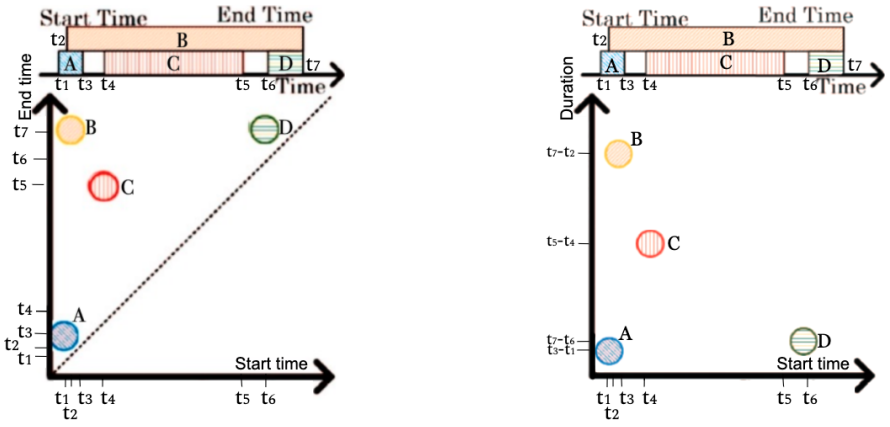
Other methods in interval-based sequence pattern mining [16, 11, 9] represent events in a manner that enables comparison between more than two events at once. This idea will be presented next.

2.4 Interval's Geometric Representation

In a geometric representation temporal intervals can be projected to points in a two-dimensional space, as in figure 2.6. At the top of both subfigure of figure 2.6 schematic representations of a series containing 4 events A, B, C and D are depicted. Below it events are projected as points in a two-dimensional space. Using start and end time as axes, figure 2.6(a) shows event `C` starting at `t4` and ending at `t5`. Figure 2.6(b) uses start and duration as axes and shows the same event `C` starting at `t4` and lasting for `t5-t4` time units.

Definition 2.4.1 *The **zero-duration diagonal** in a start-end interval geometric representation plot is a line that represents all possible locations for any temporal interval at which start and end are equal, resulting in duration values equal to zero.*

In start-end plots like Figure 2.6(a), points closer to the zero-duration diagonal correspond to intervals of lower duration than those further away. By definition, no interval's end can occur before its start, thus an interval with a negative duration is impossible, so no point in the geometric representation will be located under the zero-duration diagonal.



(a) With start time and end time as axes.[11] (b) With start time and duration as axes.[16]

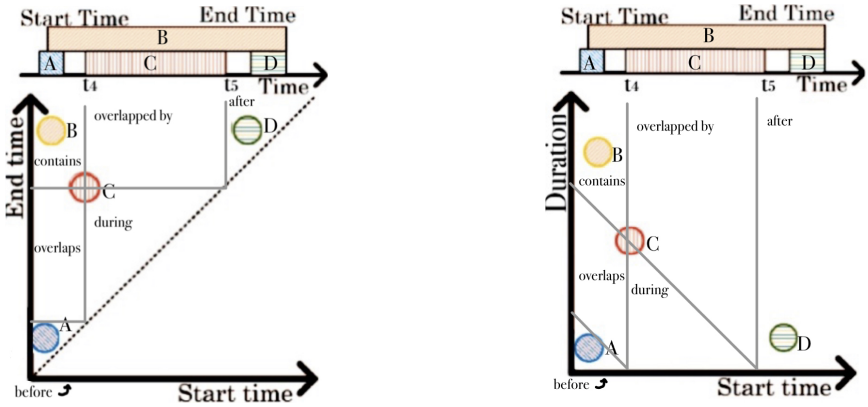
Figure 2.6: E.g. Interval geometric representation

Moreover allen's interval algebra can be projected for a given reference point into this representation drawing respective lines in [11], and [16].

Definition 2.4.2 *An **Allen line** of a reference point refers to a projected line in a interval's geometric representation separating sub spaces that mark Allen interval algebra relations of that point to others.*

To read the Allen's intervals algebra relation of a reference point to all others in a the two-dimensional space, four lines in the graph that separate subspaces corresponding to Allen's interval relations need to be taken into regard.
In the start-end plot the lines begin at the zero-duration diagonal. Two of the lines pass the reference point vertically and horizontally, representing positions of events that start and end at the same time as the reference point respectively. At the point where the vertical line meets the zero-duration diagonal, another horizontal line is drawn, representing other events that end at the time that the reference event starts, meaning the reference point meets the one on the line. Finally at the point where the horizontal line meets the zero-duration diagonal, another vertical line is drawn, representing other events that start at the moment that the reference event ends, meaning the reference point is met by the one on the line. This is shown in figure 2.7(a).

Similarly in the start-duration plot, four lines are separate the subspaces representing Allen's interval algebra relations between a reference point and the rest. Having drawn lines for the star-end plot, lines in the start-duration representation are reached by rotating the zero-duration diagonal around the graphs origin until it reaches the horizontal axis, as shown in figure 2.7(b).
Resulting figure 2.7 shows the following relations for event `C`: `A` takes place before `C`, `C` takes place before `D` and `B` contains `C`, which means that `C` takes place during `B`.



(a) With start time and end time as axes.[11] (b) With start time and duration as axes.[16]
Figure 2.7: Exemplaric Allen interval algebra relations in geometric representation

The next chapter will present how to integrate presented interval based sequence pattern mining techniques into process mining. This enables knowledge extraction from logs, containing interval events, about underlying processes as well as insights about their performance.

Chapter 3

Performance Models for Interval Events

This chapter presents the central contribution of this thesis, which is a collection of methods to gain knowledge about performance of processes containing interval events. First, section 3.1 defines interval events for process logs. Furthermore, section 3.2 explores how traces, activities and events of a process can be represented in a two-dimensional space by combining process mining and interval's geometric representation, which were described in chapter 2. Next, performance skyline in section 3.3, presents a novel method to highlight the subset of most representative events. Last, section 3.4 exemplifies how the performance skyline enables pattern discovering in processes such as trace anomaly detection. Defining conforming prescriptions for models in section 3.4.1 as well as five types of detectable anomalies in section 3.4.2.

3.1 Interval Events

Similar to events in definition 2.1.1, log traces may contain interval events. Additionally to case id, activity id and timestamp, an interval event includes a second timestamp, marking the start and the end of it.

Definition 3.1.1 *An **interval event** $e = (c, a, t^+, t^-) \epsilon \mathbb{N} \times A \times \mathbb{N} \times \mathbb{N}$ is as an tuple consisting of case id c, an activity id, a a start timestamp t^+ and an end timestamp t^-.*

Similar to a trace in definition 2.1.3, trace of interval events is ordered by its start timestamp. The subset of start and end timestamps of interval events is similar to a temporal interval as in sequential pattern mining. Thus relations between them can be described using allen's interval algebra and interval's

geometric representation as well, thus the geometrical representation of process interval events will be discussed next.

3.2 Process Geometric Representation

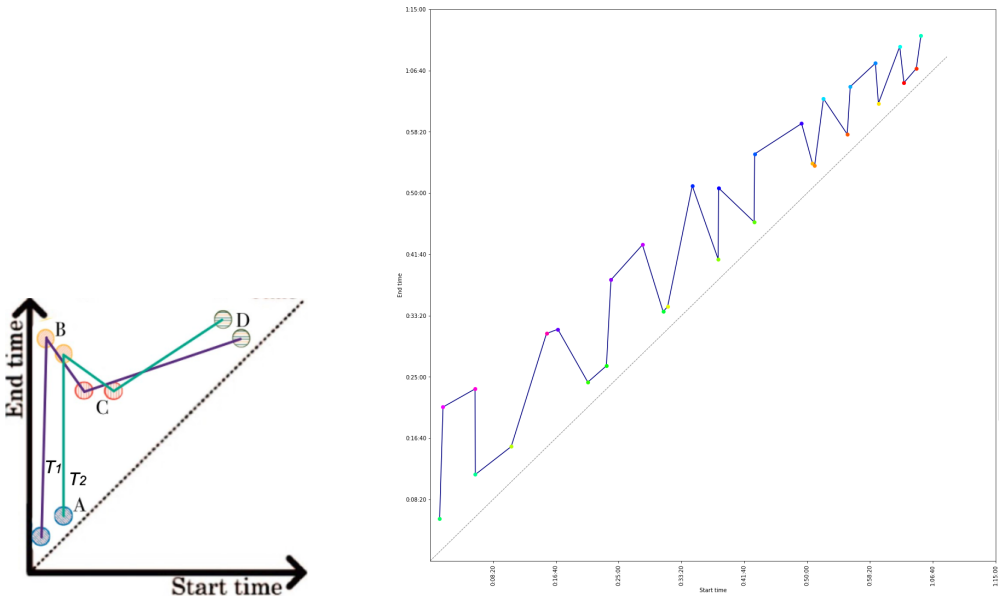
This section explains how log data from a process can be described using interval's geometric representation, based on techniques presented in section 2.4. Theoretical examples and fragments of experimental results are presented throughout this section, to explain the theoretical concepts and exemplify basic applications.

Interval events defined previously in section 3.1 contain temporal intervals. Similarly to an interval's geometric representation, process geometric representation projects an interval event to a single point in a two-dimensional space, where start time corresponds to the horizontal axis, and either end time or duration to the vertical axis. In addition, traces usually contain several interval events, and accordingly result in a collection of multiple points, as shown in example figures 3.1 and 3.2. Letters identifying temporal intervals in interval's geometric representation are used here to recollect corresponding activity ids. For real log examples, events belonging to the same activity id are projected to points of same colour. Using this representation for trace discovery enables performance knowledge extraction about the trace itself as well as all its activities.

Figure 3.1(a) describes an example of two theoretical process traces, which include four events each. An event with activity `C` in figure 3.1(a) starts at `t4` and ends at `t5`. Events for activities `A`, `B`, `C`, and `D` are connected through lines marking corresponding traces. Activity `B` events last the longest in both traces, since they are furthest away from the zero-duration diagonal. In contrast, activity `A` events appear to have the lowest duration. The first trace starts earlier and ends later than the second one. It can be identified by its longer representing line, coloured in purple. The second trace is coloured green. Events on the same vertical, like activities `A` and `B` in the second trace, start at the same time. Those sharing one horizontal position end at the same time, e.g. both activity `C` events. If a line passing through two events is parallel to the zero-duration diagonal, like activity `A` events in both traces, these events last the same amount of time.

Real logs can include many more events per trace than this theoretical example. Figure 3.1(b) depicts a trace extracted from a real log fragment. The

model proposed in this thesis discovers thirty events spread over one hour and fifteen minutes. In this case, all events have different activities. There is also a noticeable duration deviation for events following each other directly. In addition, several events appear on the same verticals, or horizontals, which means that they start respectively, at end at the same time.



(a) Theoretical example

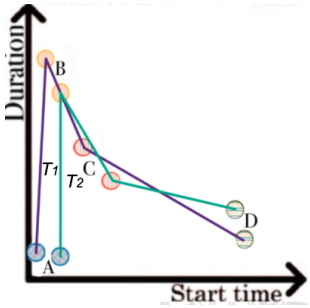
(b) Trace snippet of process called `daily` with 30 events of 30 activities spread over 01:15hrs

Figure 3.1: Process geometric representation example with start time and end time as axes.

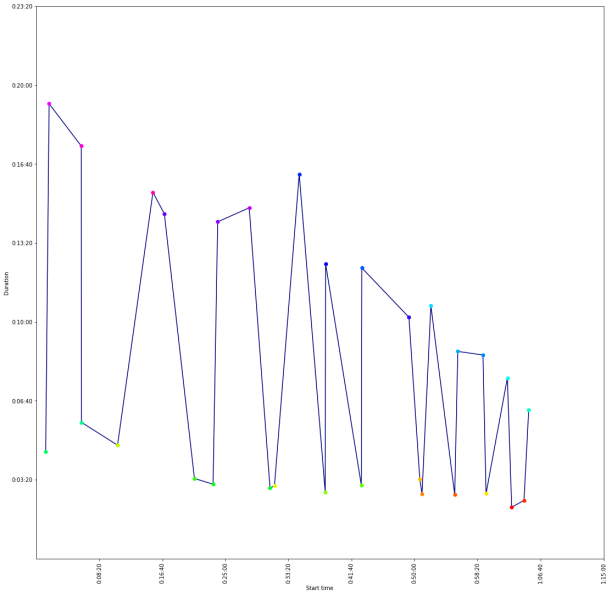
Identifying parallels to the zero-duration diagonal can be difficult for real log traces, where line segments of potential parallels are relatively short in comparison to the whole plotted trace. Therefore, an alternative way to identify events of equal duration will be described next.

Using duration rather than end time represented at the vertical axis, figure 3.2(a) shows the same activity `C` event from the first trace in figure 3.1(a), starting at `t4` and lasting for `(t5-t4)` time units. Also in this representation events on the same vertical, like activity `A`, and `B` in the second trace, start at the same time. Yet in contrast to the last representation, those sharing one horizontal position in this one bear the same duration as e.g. both activity `A` events. Consequently, if a line passing through two events is parallel to their Allen lines separating their `overlaps` and `contains` subspaces, they end at the same time, as e.g. activity `C` events in both traces.

Plotting this representation for figure 3.1(b)'s trace, figure 3.2(b) depicts the same thirty events starting within one hour seven minutes and each lasting between two and twenty minutes. Duration deviations for events following each other directly are noticeable too. The same is valid for several events appearing on the same verticals, or horizontals, meaning that they respectively start at the same time, or last at the same amount of time.



(a) Theoretical example



(b) Trace snippet of `daily` process pipeline with 30 events of 30 activities starting within 01:06hrs and each lasting 00:02hrs-00:20hrs

Figure 3.2: Process geometric representation example with start time and duration as axes.

So far, events belonging to one trace can be identified through lines connecting corresponding points. This way, multiple traces listing single event activities can be projected clearly and compared to each other, as shown in e.g. figure 3.1(a).
Visualizing all activities for all traces of a process simultaneously in a single process geometric representation plot can prove challenging because overpopulating the graphic can overwhelm the observer, overshadowing relevant information. For this reason, methods to select most relevant activities for a particular trace and process will be presented in the next section.

3.3 Performance Skyline

Named after the outline between the sky and a city's overall structure, the performance skyline is a representation of a series of interval events, which form the *critical path* of a given process trace.

The *critical path* of a trace or a process comprehends the longest series of dependent events required from start to end, which add up to the longest overall duration [17]. Consider the following example: A trace is composed of two events, `A` and `B`, both starting at `t1`. Additionally, `A` ends at `t3`, and `B` ends at `t2` with `t3>t2`. Thus, the overall duration of the trace is `(t3-t1)` and only `A` is part of the critical path. Consequently, to decrease the process duration based on this trace, event `A` needs to be sped up. Decreasing the duration of only event `B` would not improve the overall performance, it is not part of the critical path and its duration `(t2-t1)` is lower than `(t3-t1)`.

Both, the duration and the order of all events are relevant to find the critical path of a given trace. Similar to a trace, the performance skyline is also composed by a series of events, yet it only contains the subset of events held on the critical path. It is computed as follows: First, the even starting the earliest is added to the performance skyline. Next given the last added event, if there is any following event with an equal or higher start time and higher end time, it is added to the end of the skyline. This last step is repeated until there are no events left to be compared.

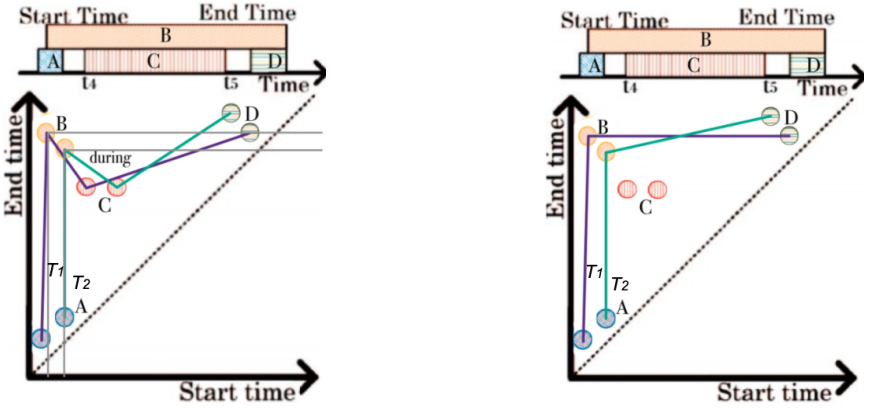
Definition 3.3.1 *The **performance skyline** ρ_c of a trace σ_c is the largest subsequence of events $\rho_c = (e_1, \cdots, e_i \ldots, e_j, \ldots, e_n)$, where $\rho_i \subseteq \sigma_c$, and $\pi_c t^-(e_i) \leq \pi_{t^-}(e_j)$ for all $1 \leq i \leq j \leq n$. Additionally, since a trace of interval events is ordered based on the start timestamps, formally $\pi_{t^+}(e_i) \leq \pi_{t^+}(e_j) \Leftrightarrow i \leq j$.*

Extending definitions of interval's geometric representation in section 2.4, the concept of *shadow* is introduced.

Definition 3.3.2 *In the context of geometric representations a reference point `A` is in the **shadow** of `B` if after drawing Allen lines for `A`, `B` is depicted in the `contains` subspace of `A`. Accordingly meaning that `A` is in the `during` subspace of `B`.*

For example in figure 3.3(a), `C` lays in `B`'s shadow.

For both process geometric representations previously presented, the performance skyline contains all points of a trace that do not lay in the shadow, as in definition 3.3.2, of any other point. An example of this is presented in figure 3.3

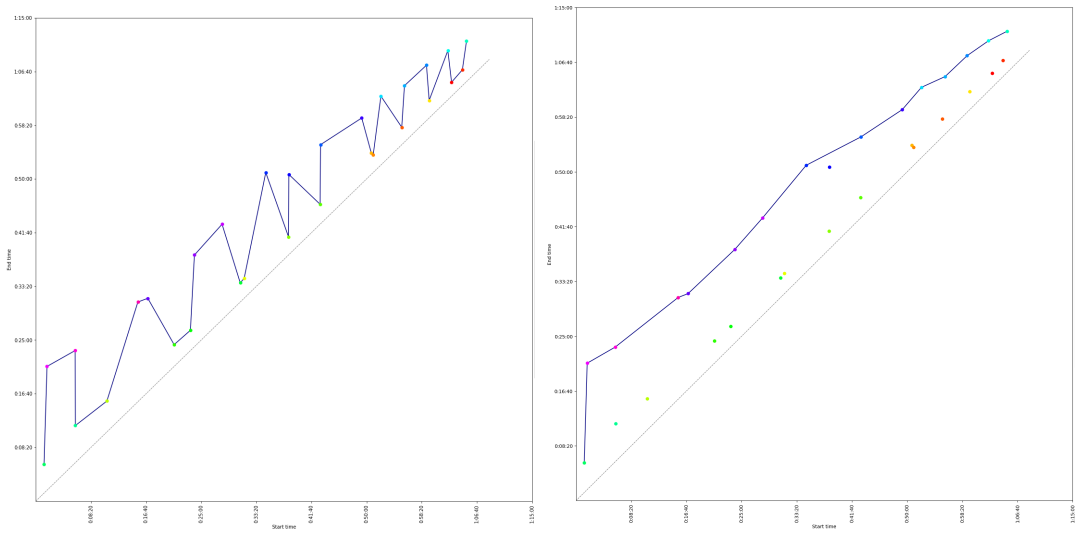


(a) Exemplary two traces with event B showing its shadow subspace (b) Performance skylines from figure 3.3(a)

Figure 3.3: Exemplary performance skyline for two traces

From the geometrical perspective, in a process geometric representation with `end time` in the vertical axis, the performance skyline connects with a line all points of a trace that are nearest to the upper left corner of the graph, as shown in figure 3.4(a). By definition, the performance skyline is increasing monotonously. The reason is that for any pair of consecutive events, if the latter ends before its prior does, only the prior will be part of the critical path contributing to the overall duration of the pair. Figure 3.4 shows the same trace previously extracted and depicted in figure 3.1(b) as well as its resulting performance skyline. Instead of showing all 30 events of the trace, figure 3.4(b) identifies 13 events that are part of the critical path for that trace.

On the other hand, considering `duration` as a vertical axis, the performance skyline connects all events of a trace that are not contained by other events. Events that contain a certain event can be found in the `contains` subspace of a interval's geometric representation plot, which results from drawing Allen lines for a reference point representing the given event, as presented in figure 3.2(a). Figure 3.5(a) shows the same trace previously extracted and depicted in figure 3.2(b) and portrays Allen lines for a given reference point. The `contains` subspace for that point includes projections of two other events. For that reason, the reference point is not part of the critical path and will not be projected into the performance skyline for that trace. Figure 3.5(b) presents the resulting skyline constituting of 13 events that are part of the critical path for



(a) Trace snippet of `daily` process with 30 events of 30 activities spread over 01:15hrs

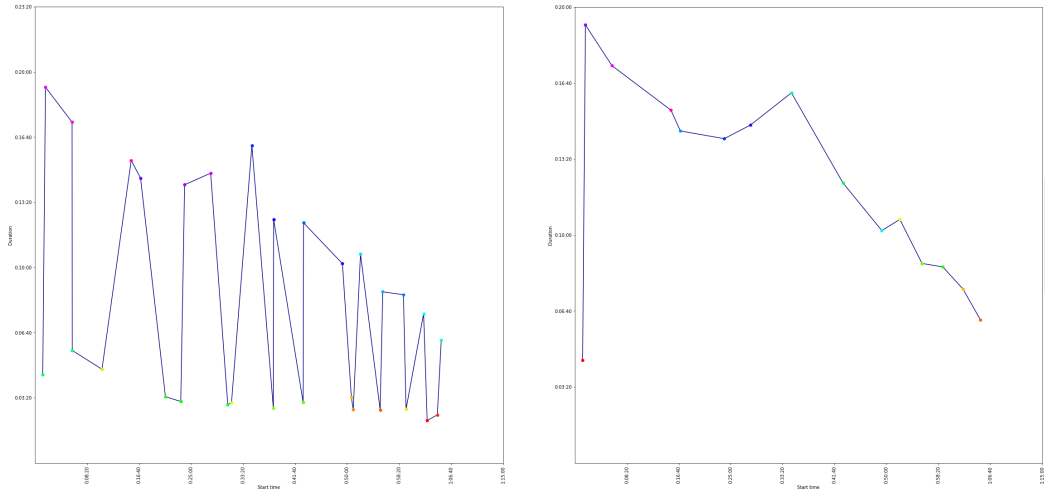
(b) Performance skyline of trace in figure 3.4(a)

Figure 3.4: Performance skyline of a real trace with start and end time as axes

the trace.

The performance skyline offers a summary of a trace by presenting its critical path. Resulting performance skylines in both start-end as well as start-duration representations offer the same amount of knowledge about traces and their critical paths. In addition, given that in a start-duration representation Allen lines need to be drawn for every point in a trace in order to sketch, as well as to understand its performance skyline, only process geometric representations with start time and end time as axes will be used for further visualizations in this thesis.

Process discovery techniques presented in this section enable selection of relevant events sparing avoidable resource usage and boosting routine performance in further steps. Additionally, they allow for new ways of conformance checking and pattern mining for performance models, which will be further discussed in section 5.2. One possible task that may be facilitated by the process geometric representation and the performance skyline is recognizing when a trace behaves unexpectedly. This will be further discussed in the following section, section 3.4.



(a) Trace snippet of `daily` process with 30 events of 30 activities starting within 01:06hrs and each lasting 00:02hrs-00:20hrs

(b) Performance skyline of trace in figure 3.5(a) example

Figure 3.5: Performance skyline of a real trace with start time and duration as axes

3.4 Trace Anomaly Detection

In process conformance checking, a non-conforming trace is a trace anomaly. Traces are compared against either a prescribed model, describing how a trace is supposed to manifest according to expectation, or against a statistically representative models extracted from a trace set. [14] Methods like *Token-based replay*[1], and *Ghionna's outlier detection* [8] integrate frequent pattern mining methods for trace anomaly detection. Most literature regards non-conforming traces from the control-flow perspective.[1], but disregard the temporal perspective, overseeing potential anomalies. Consider for example fictional traces from figure 3.3. Structurally both traces are conformant, having the same activities happening in the same order, nevertheless $T2$ depicts an event D lasting longer than expected, shown in $T1$, thus plotting a different skyline than in $T1$ and revealing this anomaly.

Nevertheless, anomaly detection in process mining is not limited to the control-flow standpoint. For example, some temporal anomaly detection methods for business processes use Bayesian networks to identify non-conforming traces regarding time values of their activities.[15] Principle limitations of these approaches are similar to those discussed in chapter 2. They mostly use a single timestamp per event, and only recognise anomalies on activity level, ignoring their context in a trace or overall process. Furthermore they only determine

presence of a non-conforming trace and their deviation intensity. Anomaly Detection methods presented in this section detect performance anomalies by comparing traces to statistically representative trace set models. For this reason way to select representative traces and skylines out of a trace set will be presented next.

3.4.1 Trace Set Models

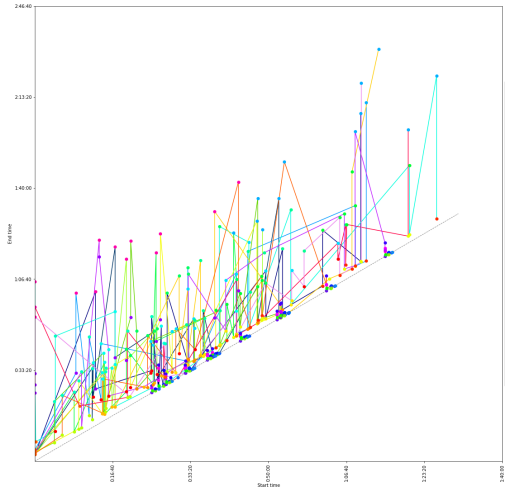
In order to compare reality with expectation, both need to be defined. Real traces used for the examples in this section were extracted from the experiments trace set, further described in section 4.1.2. Expectation definitions regarding traces and skylines will be introduced next. For the computation of statistically representative models of an event log, all are aligned to the left, so that start and end times of all events are relative to the first event's timestamps in their corresponding trace. This way averaging timestamps excludes bias for processes where traces are executed one after another. For completeness, figures originated from real data are provided as extended versions in appendix A

Figure 3.6(a) shows a sample of a real log containing 390 events describing 30 activities on 13 traces of a process. Points in similar positions and same color can support relative confidently finding a expectation trace, describing expected behaviour for future traces. But even if some recurring patterns are identified from a such figure, contemplating several traces simultaneously challenges comparison of yet another additional trace with any possible formed expectation. For this reason a representative trace to compare future traces to, will be identified.

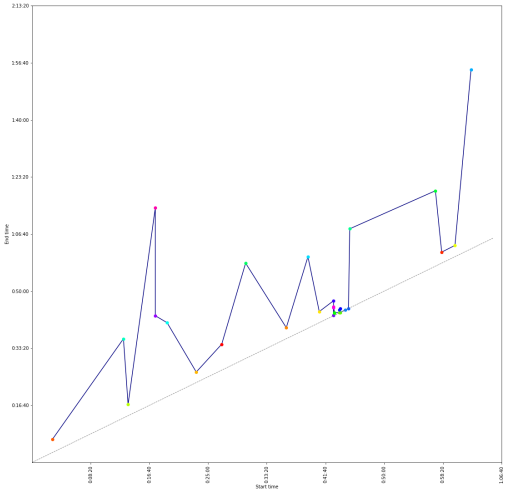
Definition 3.4.1 *The **average trace** of a process is the resulting trace of averaging all events start and end times for each activity on the trace set.*

Figure 3.6(b) shows the average trace of the previously elaborated example in figure 3.6(a). An average trace is suitable as a comparable expectation for inquires that involve all activities of a trace set. It eases the view to gain representative knowledge about all activities start and end times as well as the relationships between consecutive activities. It can be used for the detection of multiple anomaly types, further described in definitions 3.4.5 and 3.4.7.

Nevertheless, if only activities belonging to the critical path, as mentioned before in section 3.3, are of sufficient to detect an anomaly, a selection of the average trace is better suited to avoid computational overhead. Combining the average trace, as in definition 3.4.1 and the performance skyline in section 3.3, an *average trace skyline* is introduced.



(a) Real log snippet from `daily` process with 390 events of 30 activities in 13 traces starting within 01:06hrs and ending between 00:05hrs-02:00hrs



(b) Average trace from figure 3.6(a)

Figure 3.6: Average trace example

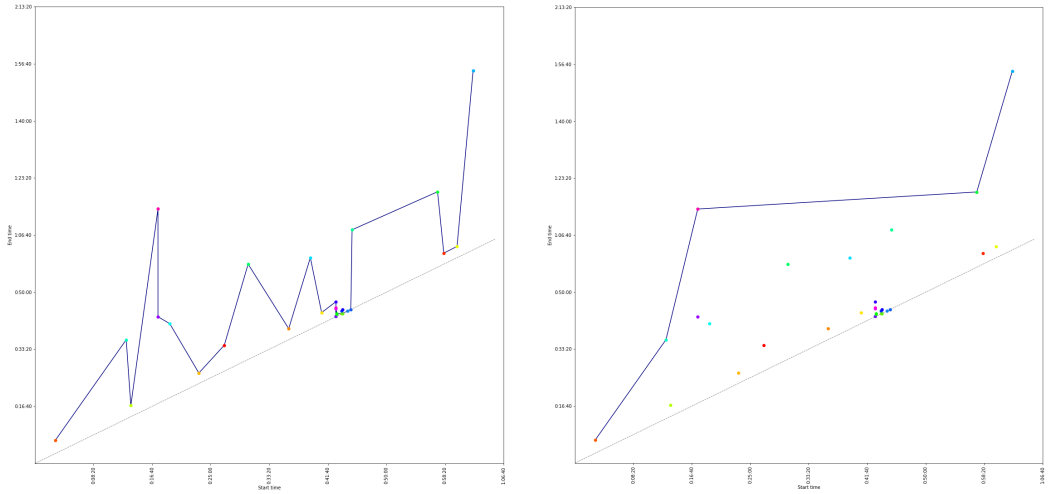
Definition 3.4.2 *The **average trace skyline** of a process is the performance skyline of the average trace of all the traces in its event log.*

Figure 3.7(b) shows the average trace skyline of the trace set depicted in figure 3.6(a), which is the skyline of figure 3.6(b), also shown in figure 3.7(a). An average trace skyline is suitable to evaluate performance expectations for a trace set because it facilitates gaining knowledge about activities that are often part of the critical path and their relations to each other. It can be used for the detection of the anomaly type presented in definition 3.4.6.

Even so, groups of traces that have different sets of activities on their skyline might be representative, and conforming, without appearing most often. For this reason the order of steps, averaging and computing a skyline, for a trace set leads to different expectation skylines.

Definition 3.4.3 *The **average skyline** of a process is the resulting skyline from averaging all skylines per activity id for all activities, which appear on any trace's skyline.*

Figure 3.8(a) shows the collection of skylines for figure 3.6(a)'s trace set. Next to it, figure 3.8(b) shows the average of the depicted skylines in figure 3.8(a), which is consequently the average skyline of traces in figure 3.6(a). An average skyline is suitable to consider performance aspects for a given trace set,



(a) Average trace from figure 3.6(a) (b) Skyline of average trace from figure 3.7(a)

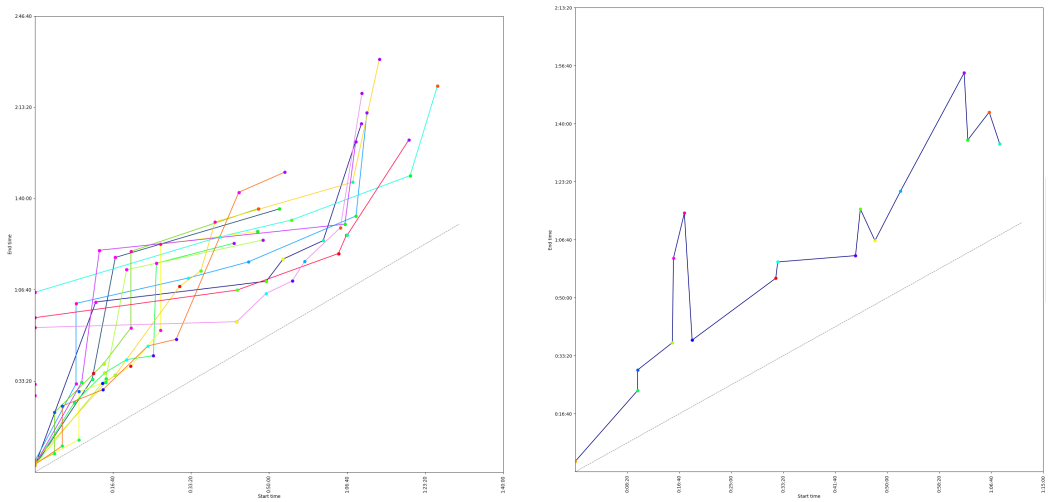
Figure 3.7: Skyline of average trace example

because offers knowledge about activities that might be part of any trace's critical path and their relations to each other. Consequently it is more inclusive regarding expectations of skyline activities. It can be used for different types of anomaly detection as in definition 3.4.6, and definition 3.4.7.

Nonetheless, adding every activity on any skyline ever to the average skyline can produce an expectation skyline that is significantly sensitive to outliers because it includes activities that might only be part of a skyline computed from an anomaly trace. As a trade off between only including most frequent critical path's activities and all activities from any critical path disregarding their relevance, an expected skyline activity set will be introduced next.

Definition 3.4.4 *The **expected skyline activity set** of a process is the set of activities that have a probability, equal or higher than a given value, of appearing on almost every skyline of that process trace set.*

For the trace set used for all other definitions, and depicted in figure 3.6(a), the probabilities distribution regarding the activity appearances on all skylines, as in figure 3.8(a), is depicted in figure 3.9. Using this figure a expected probability value shall be chosen to define the expected skyline activity set of the presented process. The higher the chosen probability value, the fewer activities will be part of the expected skyline activity set, resulting in less identified trace anomalies of high confidence. Analogously the lower the chosen probability value, the more activities will be part of the expected skyline activity



(a) Real skyline snippet from `daily` process with 390 events of 30 activities in 13 traces starting within 01:06hrs and ending between 00:05hrs-02:00hrs

(b) Average of skyline in figure 3.8(a)

Figure 3.8: Average skyline example

set, resulting in more identified trace anomalies of lower confidence.

An expected skyline activity set is suitable to consider performance aspects for a given trace set as well as the activity set. Furthermore it provides knowledge about activities that are or not part of the critical path given their appearance probability in a given skyline set. It can be used for the detection of the anomaly type presented in definition 3.4.7 by choosing a appearance probability threshold, exemplified in figure 3.9.

Using these definitions for expectation traces and skylines, anomalies can be recognized. Next section 3.4.2 will present definitions for five types of trace anomaly, exemplified and discussed in chapter 4.

3.4.2 Trace Anomaly Types

Representative traces, as in section 3.4.1 have been computed from the experiment trace set to identify anomalies. Next five types of anomaly found in a data set will be introduced. Example visualizations and corresponding possible causes leading to anomalous behaviour for each type will be further discussed in trace anomaly detection results in computational experiments and discussion.

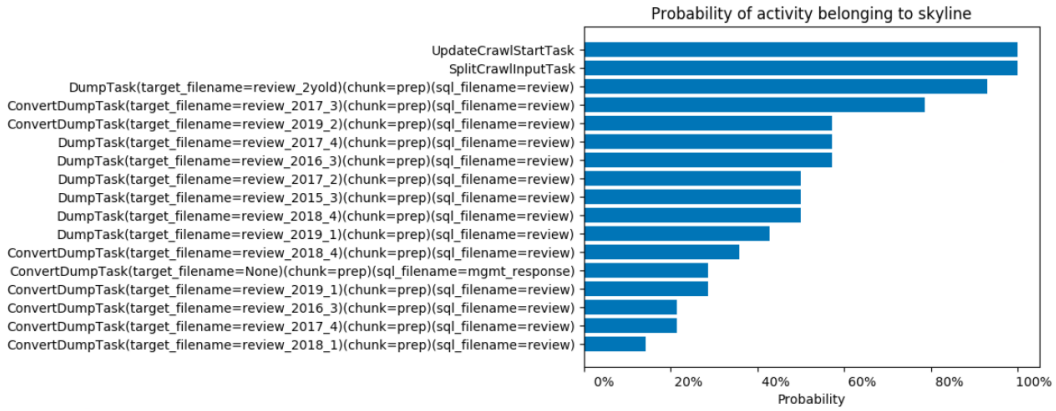


Figure 3.9: Performance skyline activity set with corresponding probabilities.

Definition 3.4.5 *A **number of events in trace anomaly** is a trace with a deviation on the expected number of events for a trace given how many events are in its average trace. An anomaly is identified if there are more or less than the expected number of events plus or minus a standard deviation, respectively.*

The performance skyline exhibits the anomaly trace, displaying the anomaly with a significant lower or higher amount of points alongside with the corresponding average trace. The significant difference of points between both traces is marked as in figure 4.5.

Furthermore, even traces that have an habitual amount of events can still contain anomalies, as presented next.

Definition 3.4.6 *A **percentage of events in skyline anomaly** is a deviation on the expected percentage of events for a trace skyline given what percentage of events are in its average trace skyline. An anomaly is identified if there are more or less than the expected number of events plus or minus a standard deviation, respectively.*

Excluding number of events in trace anomalies, the performance skyline exhibits the anomaly trace displaying the anomaly's skyline with a significantly lower or higher amount of points alongside the corresponding average trace skyline, including projections for all other point in the average trace. The significant difference of points between both skylines is shown in figure 4.6.

Furthermore, also skylines having an habitual amount of events per case can still contain anomalies, as presented next.

Definition 3.4.7 *A **skyline activity set anomaly** is a deviation of a trace, whose skyline lacks activities from its corresponding process expected set of activities.*

The performance skyline exhibits this anomaly by displaying the anomaly's trace skyline alongside the corresponding average skyline, as in definition 3.4.3, including projections for all points that present an equal or higher probability in figure 3.9 than a chosen threshold. The set of activities lacking in the anomaly's skyline can be identified by comparing the activity set missing between skylines as in e.g. figure 4.7.

Even so, traces that have an habitual set of activities on its skyline can contain anomalies, as resented next.

Definition 3.4.8 *A **number of events per activity on skyline anomaly** is a deviation in a trace, whose skyline has at least one activity with a unexpected amount of appearances. An unexpected amount of appearances for an activity in a trace qualifies as an anomaly if it overpasses the expected amount of appearances for that activity in particular on the expectation skyline plus or minus a standard deviation.*

The performance skyline in exhibits this anomaly by displaying the anomaly's skyline alongside the corresponding expected event skyline, including points of the same activity for as many events as they are expected to be in it, similar to a skyline computation. In figure 4.8 the amount of events of the same colour is compared between skylines as in figure 4.8.

But also traces that have an habitual amount of events per activity can contain anomalies, as demonstrated next.

Definition 3.4.9 *A **number of events of unexpected duration on skyline anomaly** is a deviation of a trace, whose skyline has a certain amount of activities of an unexpected duration anomaly. For this thesis a deviation of unexpected duration qualifies as an anomaly, if it lasts for more or less than the average duration for that activity plus or minus respectively, a standard deviation.*

The performance skyline exhibits this anomaly by displaying the anomaly's skyline alongside the corresponding average skyline, as in definition 3.4.3. Events duration values for same activities are compared to each other. All activities with duration anomalies tend to be next to each other and depict a similar form for both skylines. The difference is that certain segments of that

shape may appear larger or smaller, depicting the zone where the anomaly takes place. Not only unveiling the anomaly trace, but also the region, where most anomaly activities lay. For example in figure 4.9.

Having defined trace anomaly types, experiment results in trace anomaly detection results will exemplify their behaviour in performance skyline in the next chapter.

Chapter 4

Computational Experiments and Discussion

Having previously introduced performance models for interval events, this chapter discusses using the process geometric representation and the performance skyline on real interval events for process discovery and anomaly detection purposes. These models were implemented[1] and experimented on a real data set. Section 4.1 elaborates on the data set from where real interval events were extracted and on their preprocessing. Five types of anomaly found in this data set are presented and analyzed in section 4.2. Lastly, section 4.3 summarizes the findings and results of this thesis.

4.1 Data Set

This dataset comprehends three months of logs for a process called `daily` at TrustYou GmbH, a German online feedback analysis company. Broadly explained this process ingests customer reviews from multiple sources, does multiple transformations, aggregates them and stores the result at a given location. Being a data process, only computing resources are involved and thus control-flow deviations such as order of activities execution, do not vary without showing performance deviations on interdependent activities as well. Next, data processing will be discussed.

4.1.1 Data Preprocessing

Event logs often are stored in a semi-structured collection of text files containing both usable and disposable information for a specific task, as shown

[1] https://github.com/andreamalhera/performanceskyline

in figure 4.1. In order to apply a data set to a model, interval events need to be extracted from their logs and transformed into a suitable format. This procedure is called *data preprocessing*. This section accommodates preprocessing steps that are necessary for this specific dataset but are nevertheless generally applicable. Preprocessing steps for real logs include:

1. Extracting relevant lines from logs.
2. Extracting relevant columns per line.
3. Matching and merging start and end timestamps to corresponding single interval events.
4. Cleaning activity names.

Logs used in this thesis originate from data processes that are automatically generated using *Luigi*. *Luigi*[2] is a Python package that helps to build pipelines of jobs, which can be scheduled to run as resources permit. Additionally Luigi tracks jobs by generating logs of their execution. Any time the whole pipeline is executed, a new log is produced. Bearing in mind the terminology introduced in section 2.1, a log from Luigi represents a trace and contains corresponding events.

```
2019-09-01 09:29:25,787 DEBUG    worker.py:260  - Checking if ConvertDumpXmlifyTask(date=2019-
09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response) is complete
2019-09-01 09:29:26,363 INFO     worker.py:313  - Scheduled ConvertDumpXmlifyTask(date=2019-09
-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response) (PENDING)
[...]
2019-09-01 16:50:12,097 INFO     worker.py:58   - [pid 8026] Worker Worker(salt=235269763, h
ost=                , username=          , pid=45075) running    ConvertDumpXmlify-
Task(date=2019-09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_re
sponse)
2019-09-01 16:50:12,120 INFO     worker.py:336  - ConvertDumpXmlifyTask(date=2019-09-01_09-29-
01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response) is currently run by worker
Worker(salt=235269763, host=                , username=          , pid=45075)
[...]
2019-09-01 17:01:16,434 INFO     worker.py:80   - [pid 8026] Worker Worker(salt=235269763, ho
st=                 , username=          pid=45075) done    ConvertDumpXmlifyTask(d
ate=2019-09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response)
[...]
2019-09-03 03:43:12,356 INFO     worker.py:336  - RootTask(date=2019-09-01_09-29-01, prev_date
=2019-08-30_13-45-01, chunk=03) is currently run by worker Worker(salt=235269763, host=
        , username=          , pid=45075)
2019-09-03 03:43:13,140 INFO     worker.py:80   - [pid 33747] Worker Worker(salt=235269763, ho
st=                , username=          , pid=45075) done      RootTask(date=2019-09-01_09-29-
01, prev_date=2019-08-30_13-45-01, chunk=03)
2019-09-03 03:43:13,142 DEBUG    worker.py:341  - Asking scheduler for work...
2019-09-03 03:43:13,153 INFO     worker.py:332  - Done
2019-09-03 03:43:13,154 INFO     worker.py:333  - There are no more tasks to run at this time
```

Figure 4.1: Example of real event log.

[2]https://luigi.readthedocs.io/en/stable/index.html

In order to extract interval events only the lines of a log marking the start or the end of an event are relevant. In figure 4.1 these lines can be identified by looking for the ones that contain a id, which are labelled with `[pid...]` in the example above. Each of these lines contains information about either the start or end of a event.

Furthermore, as described in section 3.1, interval event elements need to be extracted. In the account of every log having a unique filename, this is used as a case id for each event. Timestamps are extracted by taking all characters before the first comma in a line, and a preliminary activity id is extracted by taking the last part of the line, after the terms `running` or `done` for this particular example. Regular expressions are used to automatise both previously mentioned extraction steps.
Assuming `example-process.2019-09-01.log` is the example log's name in figure 4.1, cleaning the extracted bold lines from the previous snippet would result in the preliminary events listed on table 4.1.

case id	preliminary activity id	timestamp
example-process. 2019-09-01	ConvertDumpXmlifyTask (date=2019-09-01_09-29-01, prev_date=2019-08-3_13-45-01, chunk=prep, what=mgmt_response)	2019-09-01 16:50:12
example-process. 2019-09-01	ConvertDumpXmlifyTask (date=2019-09-01 _09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response)	2019-09-01 17:01:16

Table 4.1: Preliminary event extraction by line for each bold line in figure 4.1

Preliminary activity ids are unique for each event because of their unique parameters. For this reason after extracting preliminary information as in table 4.1, matching events, which have equal value for both case id and activity id, are merged into a single interval event. Since preliminary activity ids are unique, only two events that will match each other. In figure 4.1, both preliminary events represented in the bold lines, containing the term `[pid...]` in the snippet, mark the start and end of the same activity, thus they are merged into a single interval activity, which is shown in table 4.2.

Unique combinations of parameters and values for each event compose unique preliminary activity ids. Nevertheless *activity names*, which refer to the part up to the first parenthesis, may match for some events that have different preliminary activity ids, as shown in table 4.3. All events listed in there share a

case id	preliminary activity id	start time	end time
example-process. 2019-09-01	ConvertDumpXmlifyTask (date=2019-09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=mgmt_response)	2019-09-01 16:50:12	2019-09-01 17:01:16

Table 4.2: Merged preliminary events results in interval event.

common activity name: `ConvertDumpXmlifyTask`. Removing all parameters would assume that none of them are important to discern different activities from each other. Instead the method described next extracts only relevant parameters for a suitable activity id.

ConvertDumpXmlifyTask(date=2019-09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep,what=mgmt_response)
ConvertDumpXmlifyTask(date=2019-09-01_09-29-01, prev_date=2019-08-30_13-45-01, chunk=prep, what=source)
ConvertDumpXmlifyTask(date=2019-09-03_03-44-01, prev_date=2019-09-01_09-29-01, chunk=prep, what=source)

Table 4.3: Examples of preliminary activity ids

The following method selects a suitable activity id form a preliminary id without specific field expertise for a dataset:
For every group of preliminary activity ids sharing the same activity name in a data set, all parameters are inspected. If a certain parameter value is equal in all preliminary events, the parameter is removed for being redundant with activity name; If a certain parameter value is equal in all preliminary events within the same log, the parameter is removed for being redundant with the case id; if a parameter value has a different value for every preliminary activity id within the data set, the parameter is removed for being redundant with the interval event itself. Note in this case that activity name is not a parameter, and thus it can not be removed. All other parameters are kept, and lead by their activity name result in the new activity id. If after this procedure no parameters are left, the activity name becomes the new activity id.

In the example, after comparing all events with activity name `ConvertDumpXmlifyTask`, preliminary activity id `ConvertDumpXmlifyTask(date=2019-09-01_09-29-01, prev_date=2019-08-3_013-45-01, chunk=prep, what=mgmt_response)` results in `ConvertDumpXmlifyTask(what=mgmt_response)`. Corresponding resulting entries are shown in table 4.4.

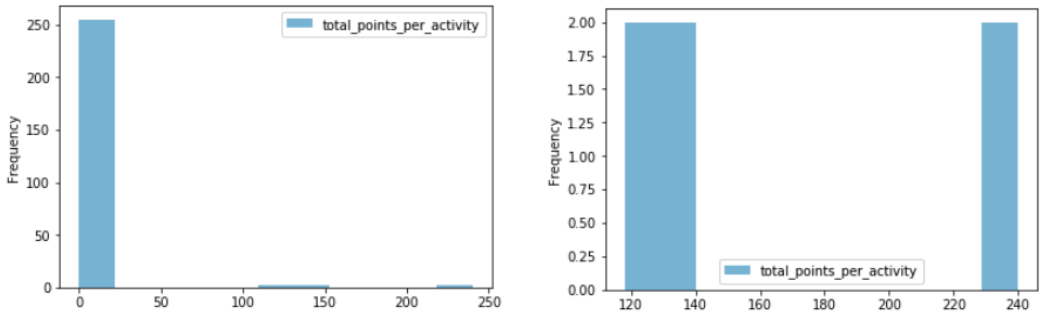
case id	activity id	start time	end time
example-process. 2019-09-01	ConvertDumpXmlifyTask (what=mgmt_response)	2019-09-01 16:50:12	2019-09-01 17:01:16
example-process. 2019-09-01	ConvertDumpXmlifyTask (what=source)	2019-09-01 14:13:56	2019-09-01 14:17:03
example-process. 2019-09-03	ConvertDumpXmlifyTask (what=source)	2019-09-03 07:33:56	2019-09-03 07:37:24

Table 4.4: Preprocessed activity ids from table 4.3

Having preprocessed log data as described in this section, now a description of its main features will be provided.

4.1.2 Data Description

The resulting collection compounds 62,074 interval events spread among 50 traces, represented by their respective `case id`. It contains 261 values for `activity id` distributed as shown in figure 4.2. A trace has on average 1238 events. Most activities appear mostly once in a trace, except for the ones listed in table 4.5, which often correspond to a few hundred events.



(a) Histogram for all activities

(b) Figure 4.2(a) augmentation for activities with high number of event points

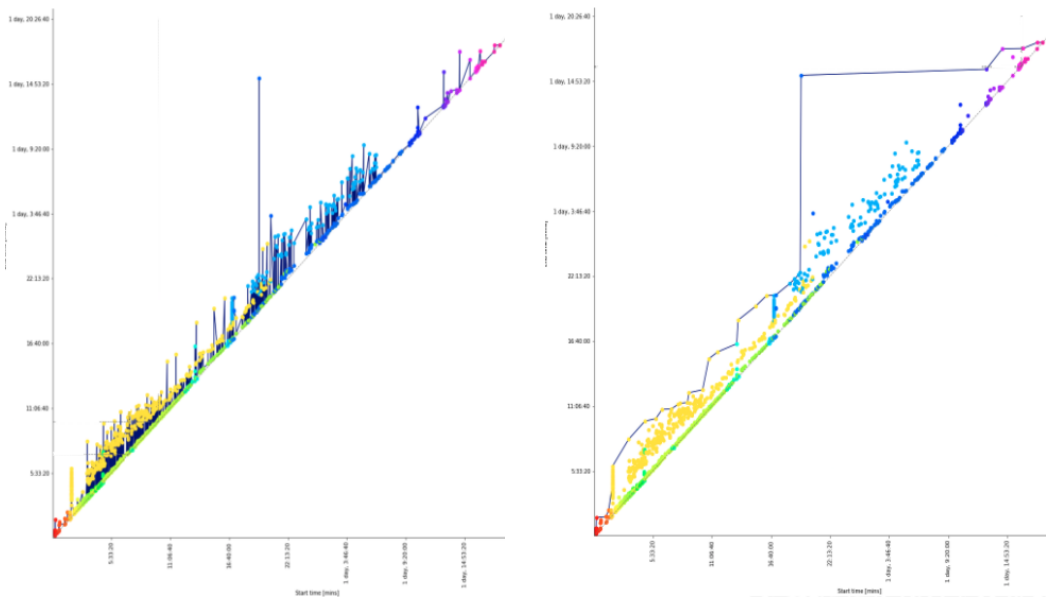
Figure 4.2: Histogram demonstrating the frequency of activities by their appearances per trace for the process `daily`.

The average trace, as in definition 3.4.1 and its performance skyline for `daily` process is depicted in figure 4.3. Each colour represents a different activity. On average, a skyline contains 65 events and 31 activities that start at some point within 42 hours and ending at some point in time between 5 minutes and 44 hours.

activity id	points per activity in trace
CrawlTask(crawler=creepy-crawly)(chunk=01)	240
ExtractTask(crawler=creepy-crawly)(chunk=01)	240
ExtractTask(crawler=creepy-crawly)(chunk=03)	139
CrawlTask(crawler=creepy-crawly)(chunk=03)	138
ExtractTask(crawler=creepy-crawly)(chunk=02)	118
CrawlTask(crawler=creepy-crawly)(chunk=02)	118

Table 4.5: Table of activities with many event points

Both visualizations show that most of the events have similar distance to the zero-duration diagonal. Nevertheless, there is one event that starts at approx. 19 hrs in relation to execution time of the trace and ends at approx. 33 hours after overall execution start. This event corresponds to a point in the process geometric representation far away from the rest of points. This results in a peak in the performance skyline, which shows how one point highly influences the critical path by dominating by duration all others that have a lower end time than this point.



(a) Trace example of `daily` process with 1238 events of 261 activities spread over 44:00hrs

(b) Performance skyline of trace in figure 4.3(a)

Figure 4.3: Performance skyline of a real trace with start and end time as axes.

As shown in figure 3.4, some activities may occur several times in the same

trace, originating multiple events of the same colour in this representation. To investigate activities of interest, specifically those with several appearances in one trace, it is suitable to plot them alone and in a way that eases comparison between them. Such a method will be discussed next.

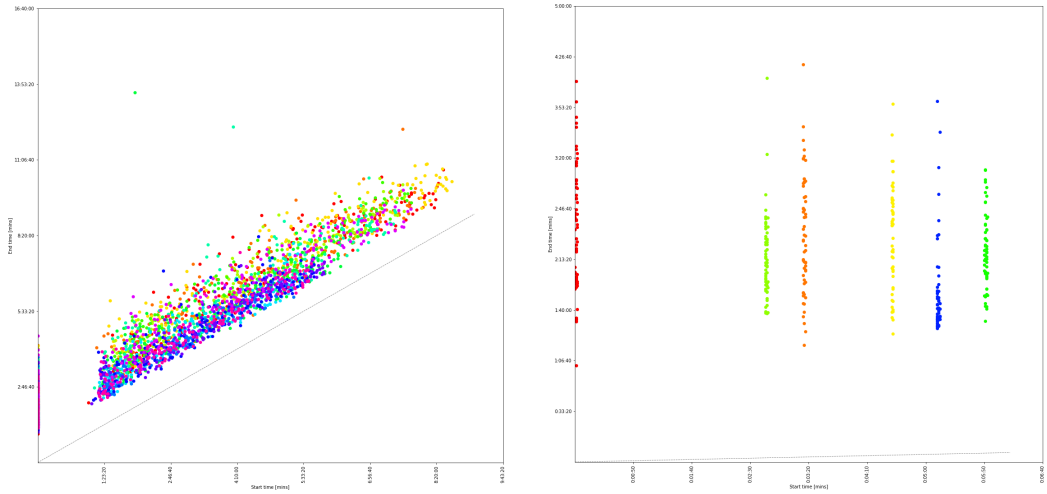
Projecting only events with the same activity id for all traces and using colours to identify the ones belonging to the same trace can ease pattern identification of activity behaviour. figure 4.4 shows some examples of this.

Figure 4.4(a) shows a snippet of a single activity composed by 3162 events on thirteen different traces. Each trace is represented by a different colour. Events timestamps variate each starting at some point within nine hours and ending at some point between one and fourteen hours. Depicted events seem to form two batches of events, as described in [10], with a noticeable gap between them.

The first batch manifests as a long vertical shape within the early minutes of start time and will be investigated further in figure 4.4(b). The gap between clusters shows no other events starting within one hour of execution.
A second batch portrays events with start times spread between one and nine hours for almost every trace. However, four of thirteen traces, identified in blue and purple shades, do not show any further events after six hours. This may indicate an anomaly for this activity in those traces. Furthermore some single events ending after eleven hours and far away from both batches may also unveil performance anomalies for specific events.

Figure 4.4(b) shows a subset of events from figure 4.4(a): A snippet of 500 events of the same activity in six different traces starting within seven minutes and lasting between one and four hours. Each trace is shown in a different colour. Depicted events on the same trace seem to all start at the same time, forming vertical long shaped clusters of events for each trace. This characteristic indicates that events of this activity run in parallel for each trace, which could point to batches and multi threading being used in that process.

Described insights of the process `daily` in this section, demonstrate how structural conformant activities on different traces may contain patterns and anomalies that are visible using the presented models. Next the results of trace anomaly detection for this trace set will be presented.



(a) Real log snippet with 3162 events of activity `CrawlTask(crawler=creepy-crawly, chunk=01)` in 13 traces starting within 08:40hrs and ending between 01:00hrs-14:00hrs

(b) Real log snippet with 500 events of activity `CrawlTask(crawler=creepy-crawly, chunk=01)` in 6 traces starting within 00:07hrs and each lasting 01:00hrs-04:00hrs

Figure 4.4: Process geometric representation grouped by activity

4.2 Trace Anomaly Detection Results

For each anomaly described in section 3.4.2 corresponding representative traces from section 3.4.1 have been computed for the presented dataset. All types of anomalies will be listed presenting a method to choose a threshold to detect anomalies without expertise knowledge about the process, visualization of an example anomaly trace and causes for such anomalies.

Visualizations in this section depict expectation traces in green. Each colored circle represents an activity that is present in the anomaly trace or in both traces. Activities that are in the expectation trace but not in the anomaly trace are plotted as red squares.

4.2.1 Number of Events in Trace

Definition 3.4.5 uses a combination of the average and one standard deviation as tolerance to identify a **number of events in trace anomaly**. A trace has on average 1238 events with standard deviation of 71.62. Figure A.10 depicts the number of events per trace in descending order for all traces as an horizontal bar chart. Drawing tolerance thresholds on this plot unveils cases that do go over it. In this case a threshold of average minus standard deviation at

1166 number of events identifies the trace `daily.2019-09-24_05-05-02` as an anomaly.

Plotting `daily.2019-09-24_05-05-02` as an anomaly performance skyline exhibits the anomaly trace, displaying the anomaly with a significant lower of points, defined by a chosen threshold, alongside the average trace. In this case for visualization purposes only a 16hrs fragment of the average trace, where there are still overlapping activities, is shown. The number of activities marked as missing activities in Figure 4.5 shows the significant difference of points between both the anomaly and average trace.

Possible causes for an anomaly with a low number of trace events are: an incomplete log, or that the process execution was interrupted at some point and it was never resumed. A possible cause for anomalies with an unexpected high number of trace events is that several tasks failed and were restarted multiple times producing several events in the log with the same activity id, and end timestamp until the process execution finished.

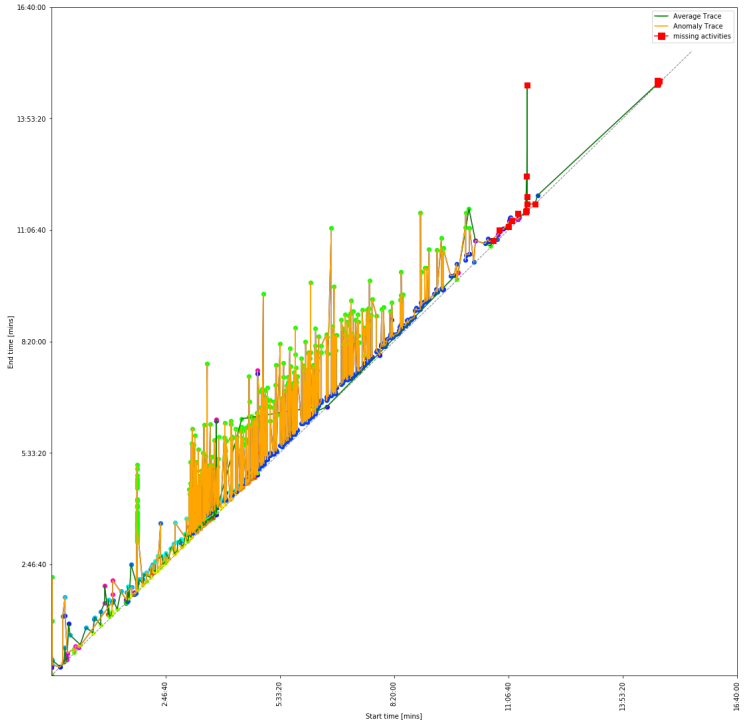


Figure 4.5: Number of events in a trace anomaly.

4.2.2 Percentage of events on skyline

Similarly to the identification of number of events in a trace anomaly, to identify a **percentage of events in skyline** a combination of the average and one standard deviation as tolerance was used (Definition 3.4.6). A trace has on average 65 events with standard deviation of 17.68. Thus for an average trace, 5.22% of the events are part of the skyline with a standard deviation of 1.38%. Multiplying the standard deviation by different factors, as shown in figure A.11, outputs different amounts and set of anomalies. Figure A.11 depicts the percentage of of events in skyline per trace in descending order for all traces as an horizontal bar chart. In this case a threshold of average minus factor 2.25 times the standard deviation results in 8.32% as `upper_bound` and 3 trace anomalies, identifying e.g. case id `daily.2019-07-29_16-03-01` as an anomaly.

factor	**avg**	**std**	**lower_bound**	**upper_bound**	**num_of_anomalies**
1.0	5.22	1.38	3.84	6.6	11
1.5	5.22	1.38	3.15	7.29	4
2.25	5.22	1.38	2.12	8.32	3
3.375	5.22	1.38	0.56	9.88	0
5.0625	5.22	1.38	0.0	12.21	0

Table 4.6: Multiplying factors and resulting number of anomalies

The performance skyline exhibits, in figure 4.6 `daily.2019-09-24_05-05-02` the anomaly's skyline with a significantly lower or higher amount of points alongside the corresponding average trace skyline, including projections for all other point in the average trace. In figure 4.6 the significant difference of points between both skylines is shown by the amount of activities on the anomaly skyline connecting lines to corresponding events that are not in the average trace skyline.

A possible cause for this behaviour is that some activities on the critical path of the process lasting shorter or longer than expected pushing dependant activities back so that they are unexpectedly part of the skyline for that trace.

4.2.3 Skyline Activity Set

Furthermore definition 3.4.7 uses the probability of an activity of being in the skyline to identify a **skyline activity set anomaly**. The complete bar chart of all activities by their probability of appearing in the skyline can be found

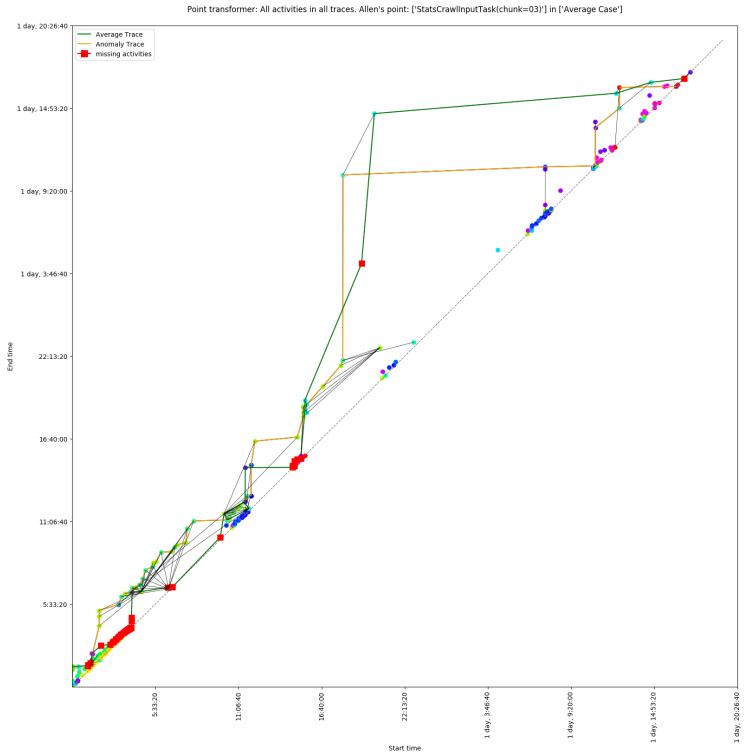


Figure 4.6: Percentage of events in skyline anomaly

in figure A.12. For this example and visualization reasons, the bar chart in figure 3.9 from section 3.4.1 will be used.

After choosing e.g. 40% as threshold and identifying `daily.2019-09-01_09-29-01` as an anomaly, the performance skyline exhibits this anomaly by displaying its skyline alongside the corresponding average skyline, as in definition 3.4.3, including projections for all points that present a probability equal or higher than 40% in figure 3.9. Activities that appear in both skylines are connected through lines between both skylines. In figure 4.7 the set of activities not present in the anomaly's skyline can be identified by comparing the activity set missing lines between skylines and listed activities on the legend.

A cause for this anomalous behaviour is that some activities that normally would be on the critical path of the process last shorter or longer than expected pushing dependant activities start time into their shadow, as defined in 3.3.2.

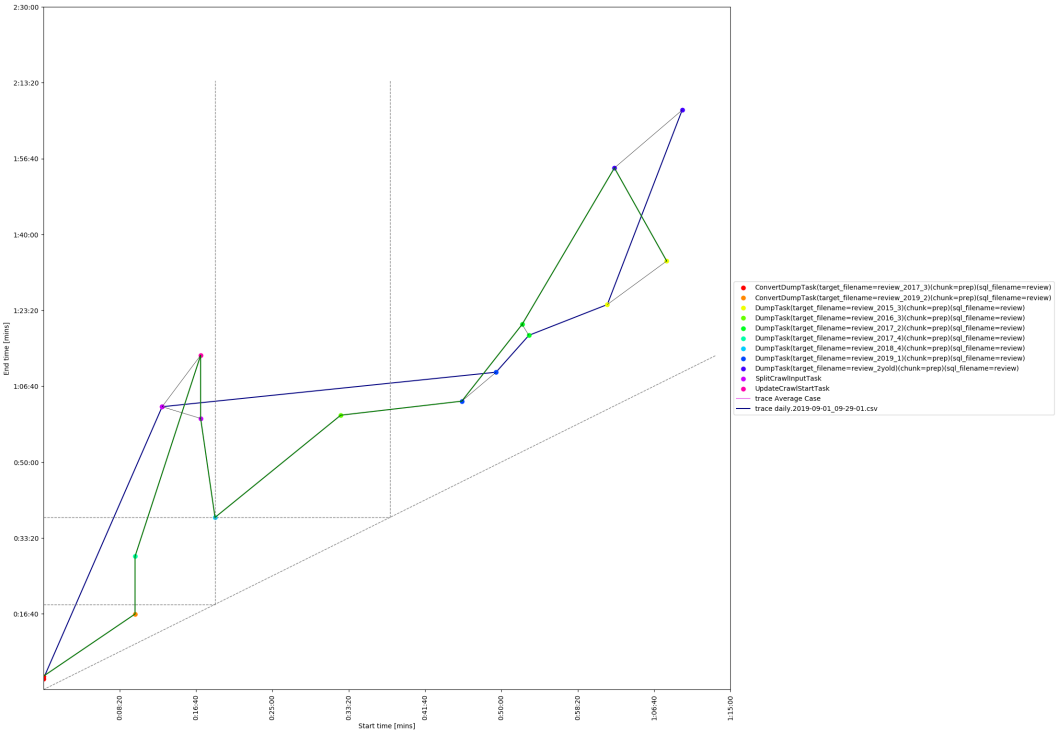


Figure 4.7: Skyline activity set anomaly example

4.2.4 Number of Events per Activity on Skyline

Additionally definition 3.4.8 uses a set of activities with a deviating amount of appearances in all skylines to identify **number of events per activity on skyline anomaly**. Figure A.13 shows a bar chart with the set of activities that have some deviation on the number of skyline appearances at all on the vertical axis by the amount of skyline missing them.

The performance skyline in figure 4.8 exhibits the anomaly of case `daily.2019-09-11_05-40-02` by displaying the anomaly's skyline alongside the corresponding expected event skyline, including points of the same activity for as many events as they are expected to be in it, similar to a skyline computation. In figure 4.8 the amount of events of the same colour is compared between skylines. If the number of events is significantly different for any activity, its representing colour is listed on the legend, so that it can be recognised.

A cause for this behaviour is that an event for a certain activity takes significantly longer to end, rising on its vertical and taking several events of the same sort into its shadow.

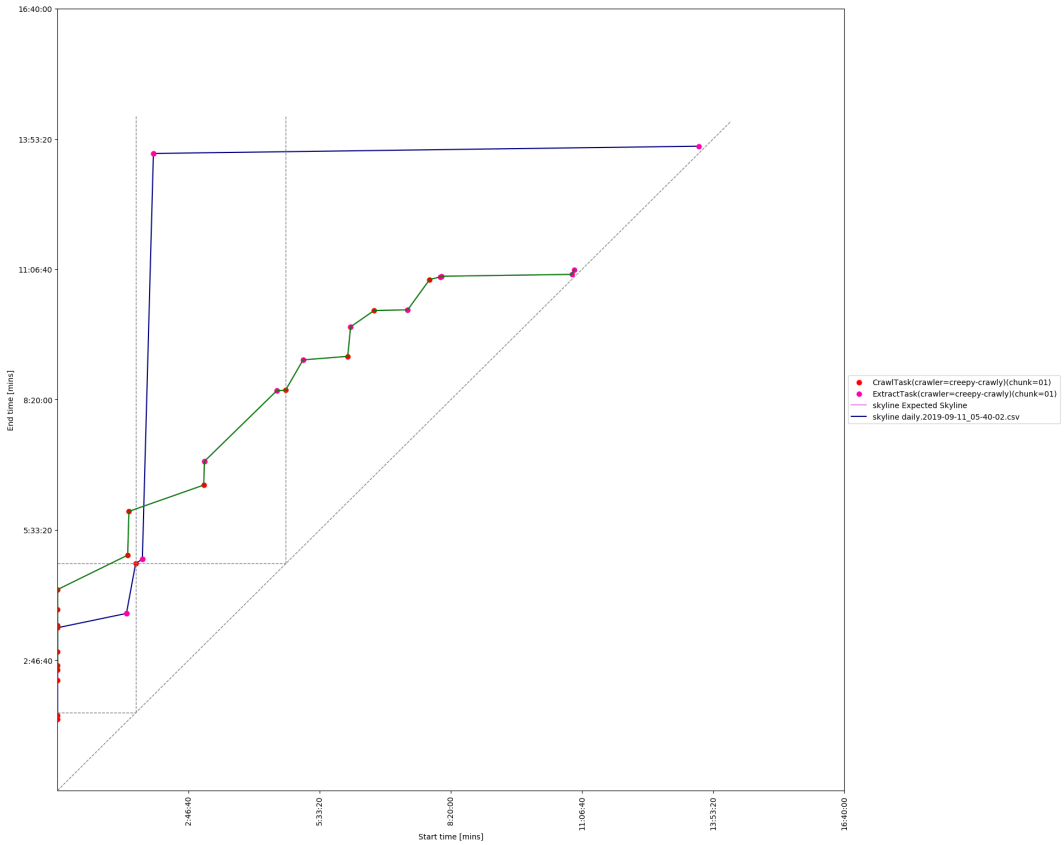


Figure 4.8: Number of events per activity in skyline anomaly

4.2.5 Number of Events of Unexpected Duration

Finally a **number of events of unexpected duration on skyline anomaly** uses the average duration per activity to identify traces containing activities that deviate over passing the set tolerance, which is in this case the average duration for that activity plus or minus respectively, a standard deviation.

The performance skyline in figure 4.9 exhibits `daily.2019-09-03_03-44-01`'s anomaly by displaying the anomaly's skyline alongside the corresponding average skyline, as in definition 3.4.3. In figure 4.9 events duration values for same activities are compared to each other. All activities with duration anomalies here are close to each other and depict generally a similar form for both skylines with the same number of points positioned in a similar relative distance and direction from each other. The difference is that certain segments of that shape, may be larger or smaller, also unveiling the time interval where the anomalous behaviour takes place.

A cause for this behaviour is that all events form an activity and additionally all events on dependant activities on that trace take longer than usual, e.g because their events process unusual big partitions of data. Nevertheless keeping relations to each other.

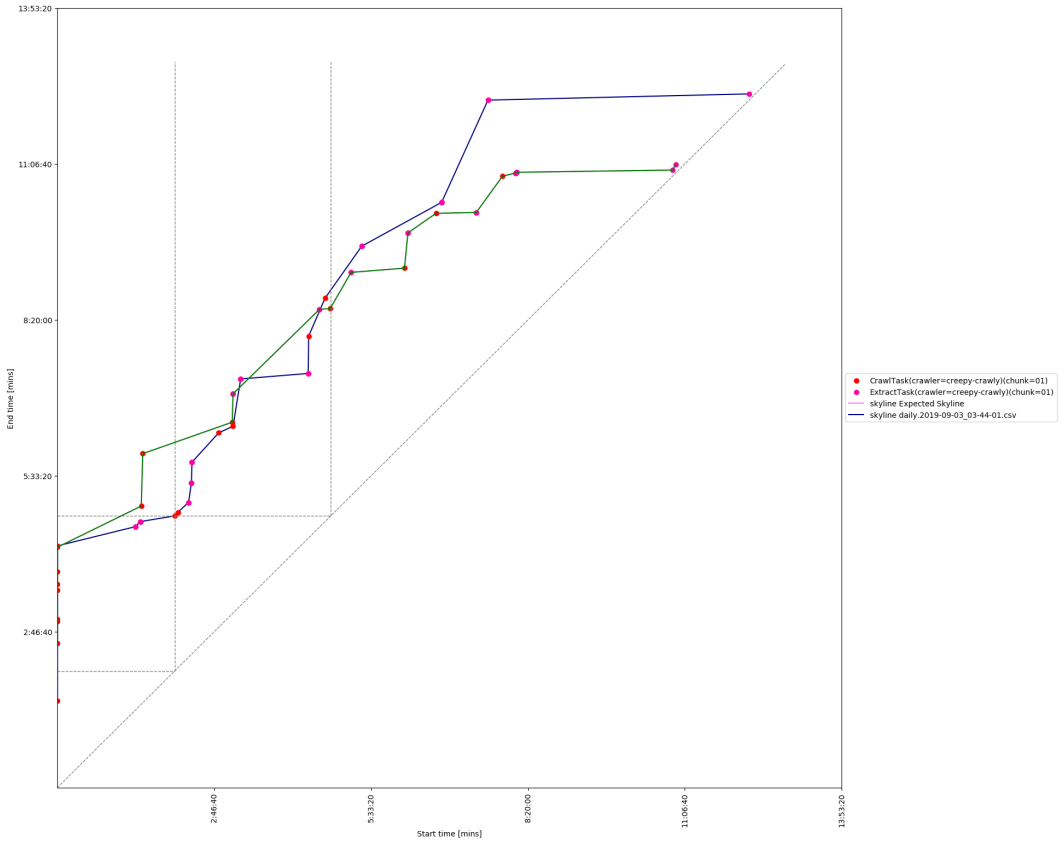


Figure 4.9: Expected activity duration anomaly

Next section will summarize findings and evaluate results of this thesis.

4.3 Discussion and Evaluation

This section evaluates results and discusses findings of this thesis presented methods for process discovery and conformance checking tasks. Section 4.3.1 outlines evaluation criteria presented in [10] and [7] for process discovery models and compares against other methods discussed in chapter 2, background knowledge and related work. Section 4.3.2 evaluates trace anomaly detection results to proof suitability of these methods for process conformance checking tasks.

4.3.1 Process Discovery

Process discovery techniques such as the α-miner's in section 2.1 describe an event log by deriving a model that covers all traces, as in figure 2.2. Also, process performance models, specifically performance skyline and trace set models, show an abstraction of all traces in a event log; however, their focus is on process performance insights rather than on control-flow. Even if the exact same metrics for process models might not be fully applicable to non model-based performance mining, following general evaluation criteria apply for both. Process models are often evaluated by how they balance four quality dimensions: *fitness*, *simplicity*, *precision*, and *generalization*. [10]

Fitness means that the discovered model should allow for the behavior seen in the event log. Since both process geometric representation, and performance skyline projects all interval events in the event log and all Allen's interval algebra relations as well, all possible behaviour for interval events in the log is allowed and represented in the model.

Simplicity states that the model should be as simple as possible to explain what is observed in the data set.
Interval events in the presented models are projected into points and traces connect those points with lines. Showing too many traces at once on the process geometric representation could result in a model lacking simplicity. Nevertheless, methods to choose trace set models in section 3.4.1 have been introduced to minimize possible overhead for observing the trace set as a whole.

Generalization means that the discovered model should generalize the example behavior seen in the event log, the model should not overfit the log.
Besides affecting simplicity, displaying all traces in an event log at once would also overfit the event log. To avoid this, performance skyline and trace set models were introduced in chapter 3. The performance skyline of a trace does

not overfit its events because it only displays the events on the critical path. Regarding overfitting for the performance skyline of a process as a whole depends on the choice of involved trace set models . Furthermore trace set models generalize start times, end times and duration for all traces by grouping all events in some way, which usually avoids overfitting.

Precision constraints that the discovered model should not allow for behavior completely unrelated to what was seen in the event log, the model should not underfit the log.
The Performance skyline does not underfit the trace because it shows by definition all events on the critical path. Regarding underfitting for the performance skyline of a process as a whole depends on the choice of involved trace set models. Trace set models averaging over activities could underfit the event log for activities with multiple appearances. If these deviate much in start time and or end time, averaging overall leads to a lack of precision for events in those dimensions. One way to avoid this for activities that appear a similar amount of times in a single trace, is to consider the average number of events per activity and create as many buckets. Then to distribute events for each activity into these buckets, and average events per bucket instead. Nevertheless, this method only works if the deviation of events per activity is not to high. For such cases, choosing to project all traces in the space is the most precise way, nonetheless on costs of generalization.

In summary, overall presented performance models offer flexibility to balance these criteria as needed, using either the geometrical representation of its traces, the performance skyline or a configuration of presented trace set models. Still needing some precision and generalization trade off limitations for event logs containing highly deviating number of events per activity.

Furthermore process geometric representation and performance skyline are methods to generate process models. Methods can be evaluated by their contribution and by comparing them to approaches that provide similar knowledge. As discussed in chapter 2, existing methods in process mining an performance mining either lack capabilities for handling interval events or require previous expertise about the process, hence comparison between them is limited. In this case, there is no much comparison since not many others have looked into performance models for interval events. Methods introduced by this thesis offer novel insights for processes performance with interval events.

Other performance mining approaches are evaluated by how suitable they are for assisting to solve a specific task [10]. Figure 4.4(b), and figure 4.3(b)

demonstrate how process geometric representation and performance skyline can be used for multiple process mining tasks, like batch detection or clustering. However the chosen task for suitability evaluation of this thesis method's is process conformance checking's trace anomaly detection.

4.3.2 Conformance Checking

In process conformance checking, a non-conforming trace is a trace anomaly. As for trace anomaly detection in the presented model, traces are compared against statistically representative models from event logs.

Having labeled anomaly traces for testing, the resulting anomaly detection classifier can be evaluated using the confusion matrix scheme shown in table 4.7. In this matrix, a trace can be either one of two classes: `positive`, meaning it is an anomaly, or `negative`, meaning it is not. Each row of the table represents the instances in a predicted class and each column represents the instances of the actual class. The contents of the table are: True positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN). Additionally the last row contains actual positives (P) with $P = TP + FN$ and actual negatives N, with $N = FP + TN$. The last column contains the sums for every row. Resulting in the last row in the last column representing 100% of the data $P+N$. With these quality metrics, the following performance measures can be computed: Precision ($\frac{\sum TP}{\sum TP+FP}$), recall ($\frac{\sum TP}{\sum P}$) and F_1-score ($2 \cdot \frac{precision \cdot recall}{precision+recall}$). The closer the F_1-score reaches to 1, the better precision and recall [12]

		Actual class		
		positive	negative	
Predicted class	positive	TP ($\frac{TP}{P+N}$)	FP ($\frac{FN}{P+N}$)	TP+FP ($\frac{TP+FP}{P+N}$)
	negative	FN ($\frac{FN}{P+N}$)	TN ($\frac{TN}{P+N}$)	FN+TN ($\frac{FN+TN}{P+N}$)
		P ($\frac{P}{P+N}$)	N ($\frac{N}{P+N}$)	P+N ($\frac{P+N}{P+N} = 100\%$)

Table 4.7: Confusion matrix scheme

Considering experiment results for anomalies of type `skyline activity set` with a tolerance threshold of >80% `of activities of the expected set are found` results in 17 predicted anomalies, as in table 4.9. Assuming that for that case labeled data shows that there are 11 actual anomalies out of the 17 predicted, and that the model would have failed to identify 4 actual anomalies, the computation of the confusion matrix following Confusion matrix scheme outcomes 4.8 and the following performance values. Precision ($\frac{11}{17} = 0.647$), Recall ($\frac{11}{15} = 0.733$), and F_1-score ($2 \cdot \frac{0.647 \cdot 0.733}{0.647+0.733} = 0.687$).

		Actual class: positive	Actual class: negative	
Predicted class	positive	11 (22%) TP	6 (12%) FP	17 (34%) TP+FP
	negative	4 (8%) FN	29 (58%) TN	33 (66%) FN+TN
		15 (22%) P	35 (78%) N	50 (100%) P+N

Table 4.8: Example computation for confusion matrix

Often, data are not labeled and thus a ground truth is missing for performance metrics. In this case a threshold can be chosen for the data set with the help of domain expertise and experiments. Different amounts of anomalous traces are detected using different tolerance thresholds.

One way to choose a threshold is as proposed in section 4.2.2. By experimenting with different thresholds, results of anomaly detection from experiments in the last section yield to table A.1, but for visualization purposes table 4.9 shows a short summary of the results. Tolerance thresholds and corresponding number of anomalies highlighted in a bold font were considered the most reasonable choices for each anomaly type. This decision was made since the number of anomalies seemed to change drastically between them and tolerance threshold in the row above, also to be observed in corresponding bar charts like figure A.10. For the anomaly type number of events of unexpected duration, thresholds are activity specific resulting in data set specific results, which are presented in section 4.2.5.

Performance tolerance thresholds might depend on processes domains, and thus precision and recall are also highly dependant on it. Even though presented models alone do not deliver a anomaly detection algorithm itself, visualizing and comparing several traces with their trace set models enables exploring choices for suitable tolerance thresholds and comparing findings with domain expertise to recognize significant anomalies.

Overall converting event log data into the performance skyline has shown to enable trace anomaly identification from business process event logs without additional information about resources or other prior knowledge. By grouping all observations into an expected trace or skyline, detailed performance characteristics can be derived revealing strong performance variations on real traces. Even in the presence of overlapping non anomalous behaviour, trace anomalies of several categories can be detected robustly because models allow for different

Anomaly Trace Type	**Tolerance Threshold**		**Detected Anomalies**
	Average	Deviation	
Number of Events in Trace	1238	**±1*71.62**	**1**
Percentage of Events in Skyline	5.22	±1*1.38 **±1.5*1.38** ±2.25*1.38 ±3.375*1.38	11 **4** 3 0
Skyline Activity Set	>70% act. of exp. set are found **>75%** >80% >85% >90%		33 **17** 11 8 5
Number of Events per Activity on Skyline	>90% anomalous act. are found **>80%** >70% >60% >40%		10 **3** 2 2 1

Table 4.9: Short summary of trace anomaly detection results for `daily` process

events in the same trace to be projected independently. Additionally, besides anomaly presence and intensity as detected in control-flow approaches like [1], proposed models in this thesis offer the novelty of categorizing anomalies in five different types.

Chapter 5

Conclusion

The research question underlying this thesis was: How to extract knowledge and model the performance of processes containing interval events? The concept and visualization of performance skylines and other trace set models, as presented in this thesis, facilitates more accurate process discovery by integrating additional performance knowledge, and enables more extensive conformance checking, detecting and discerning patterns like trace anomaly detection. It also enables potential for future process enhancement methods for processes containing interval events.

5.1 Achievements

This master thesis introduced a novel approach for performance mining of events containing multiple timestamps, combining interval-based sequence pattern mining and process mining techniques.
Afterwards, the detection of anomalous process executions in the mined performance skyline was discussed. Subsequently, process conformance was demonstrated by comparing process traces and skylines with trace set modelss on real data. Five kinds of anomaly categories were considered in the application of process geometric representation and performance skyline.
Afterwards, this model was evaluated on the task of anomaly detection. Due to its fitness, simplicity, flexibility regarding the trade-off between precision and generalization and additional dimensions of information regarding identified anomalies, this model achieved a suitable result, providing a novel solution to extract knowledge and model performance for interval events.
Even though the performance skyline trace anomaly detection methods were evaluated in the narrow scope of real process logs from TrustYou, these methods are suitable to identify trace anomalies for any kind of process with available logs containing interval events: `case id`, `activity id`, `start time`, and

`end time`.

Conclusively, the best choice for extracting the most performance knowledge from processes containing interval events is currently the performance skyline.

5.2 Future Work

Besides current limitations mentioned in section 4.3, both the process geometric representation and the performance skyline have potential to be extended as well as to be further investigated for additional tasks in process discovery, conformance checking, and process enhancement. Some ideas for expansion will be briefly discussed in this section.

First, for broader process discovery, extending the model by adding more information or enriching further the visualization of already present items; more variations of this model, as adding a third dimension plotting information about resources, dependencies or even `case id` as in dotted chart [14] could be investigated. Also reviewing variations of the models, e.g. not aligning traces to the left to gain knowledge about relations between traces, like batch processing, comparing and clustering process models regarding their interval events performance, or inferring models from streaming interval events could be researched.

Second, considering other conformance checking recognition and prediction tasks: Like further anomaly detection on activity or event level, e.g. comparing events to their neighbours, drift recognition, or predictive process monitoring for interval events, using e.g. skyline expectation maximization for performance prediction on event level.

Last, in the field of process enhancement, recognizing bottlenecks, or using detected pattern knowledge for optimal networks queuing and resource allocation efficiently could also be further investigated.

Appendix A

Long Figures and Tables

Anomaly Trace Type	Tolerance Threshold		Number of Anomaly Traces	Detected Anomalies
	Average	Deviation		
Number of Events in Trace	1238	±**1*71.62**	**1**	['daily.2019-09-24_05-05-02']
		±1.5*71.62	1	
		±2.25*71.62	1	
		±3.375*71.62	1	
Percentage of Events in Skyline	5.22	±1*1.38	11	['daily.2019-07-03_00-21-01', 'daily.2019-07-29_16-03-01', 'daily.2019-09-06_20-34-01', 'daily.2019-09-09_01-34-02']
		±**1.5*1.38**	**4**	
		±2.25*1.38	3	
		±3.375*1.38	0	
Skyline Activity Set		>70% activities of overall set are found	33	['daily.2019-07-03_00-21-01', 'daily.2019-07-15_07-34-01', 'daily.2019-09-04_23-23-01', 'daily.2019-09-20_12-49-01', 'daily.2019-09-24_05-05-02', ...]
		>75%	**17**	
		>80%	11	
		>85%	8	
		>90%	5	
Number of Events per Activity on Skyline		>90% anomalous activities are found	10	['daily.2019-09-04_23-23-01', 'daily.2019-08-25_03-39-01']
		>80%	**3**	
		>70%	2	
		>60%	2	
		>40%	1	

Table A.1: Long summary of trace anomaly detection results for `daily` process

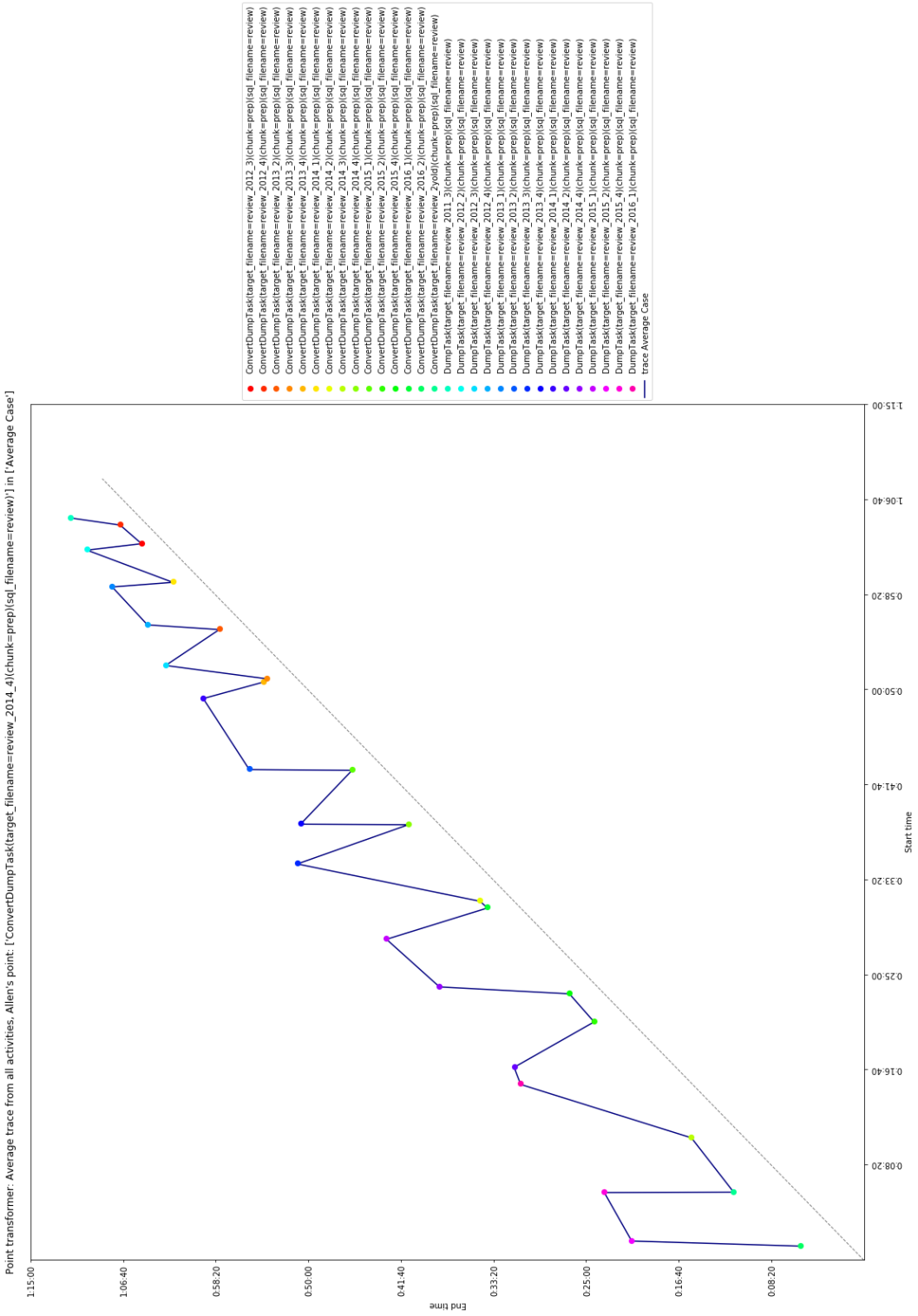


Figure A.1: Extended: figure 3.1(b)

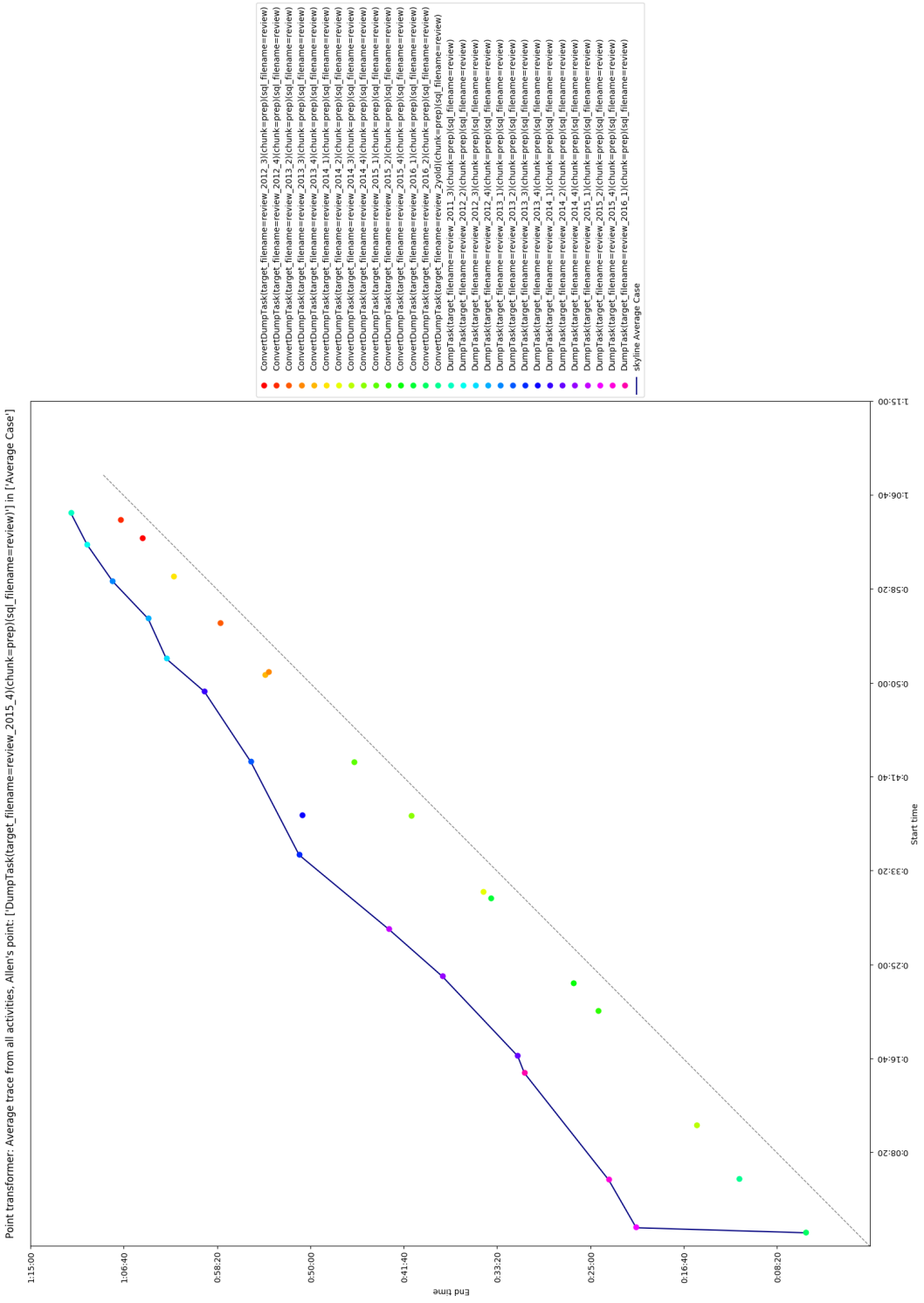


Figure A.2: Extended: figure 3.4

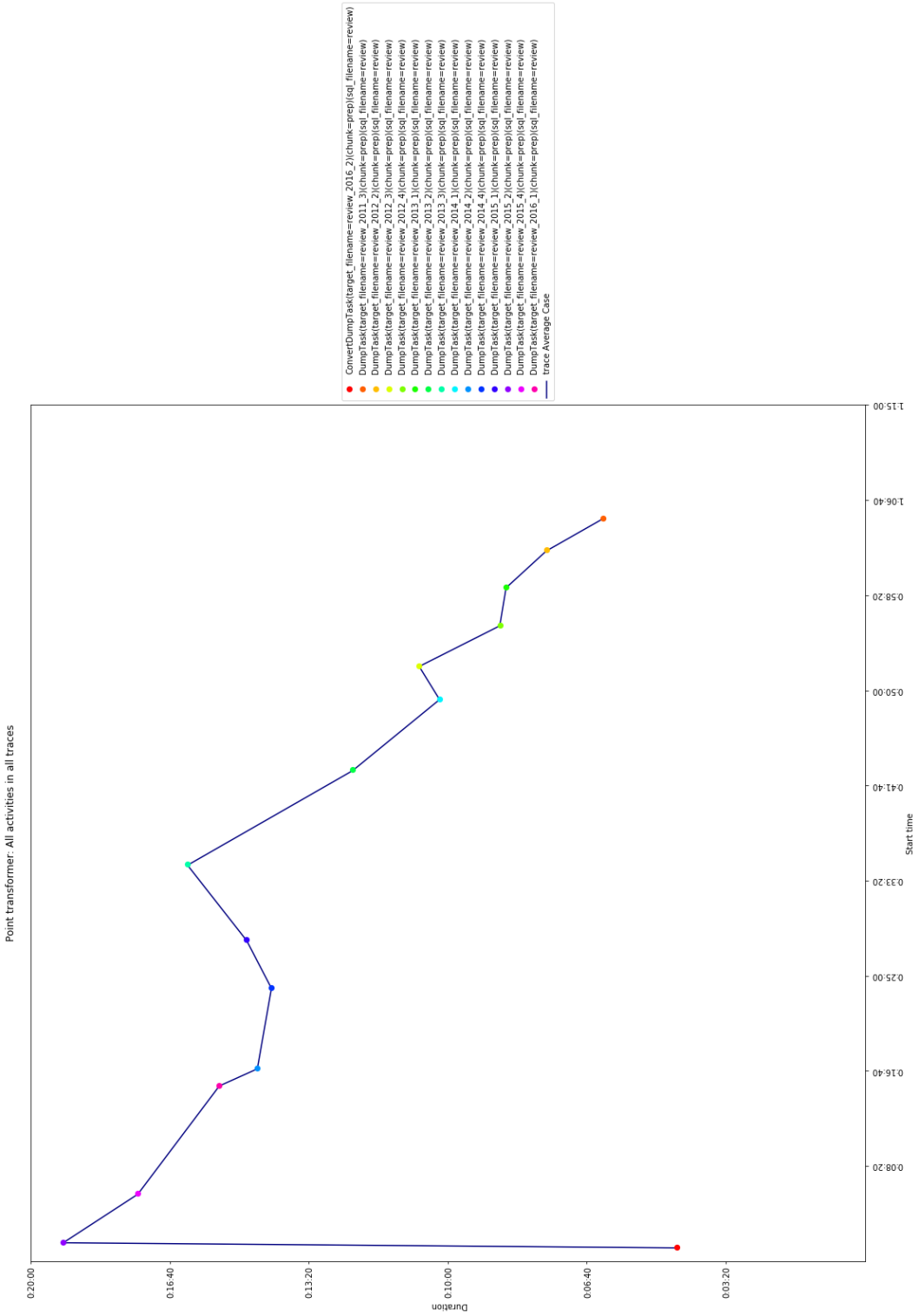


Figure A.3: Extended: figure 3.5(b)

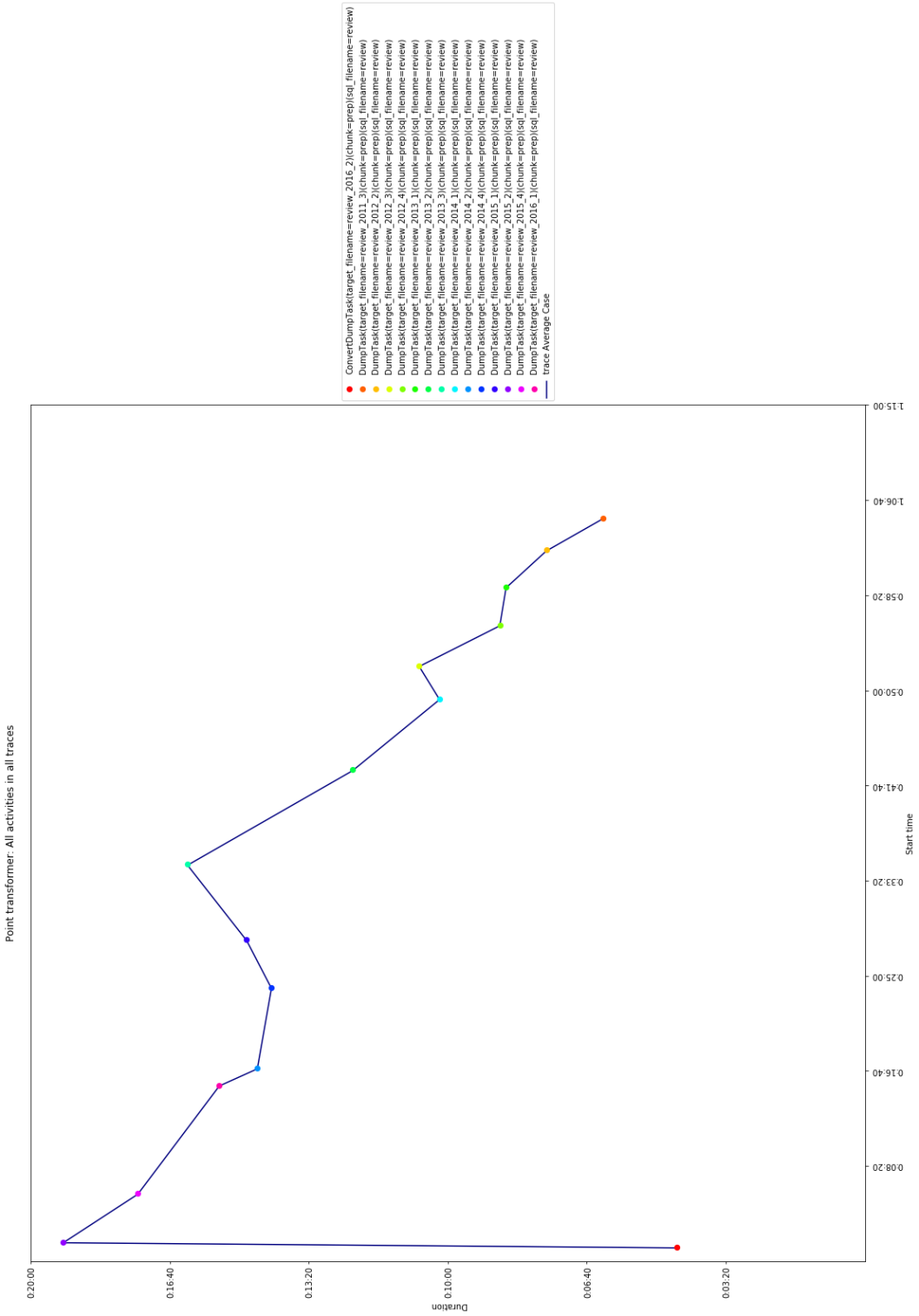


Figure A.4: Extended: figure 3.5(b)

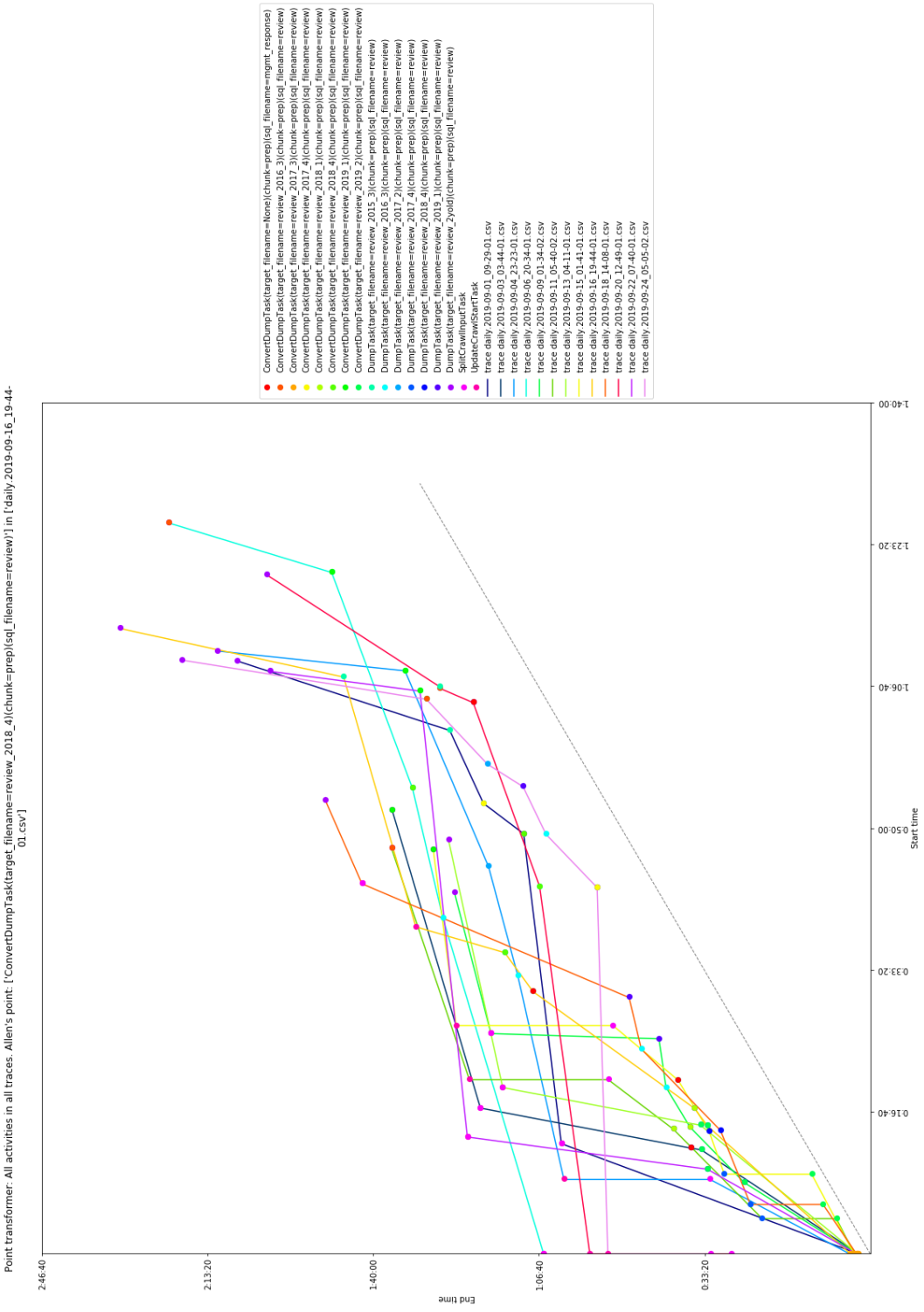


Figure A.5: Extended: figure 3.8(a)

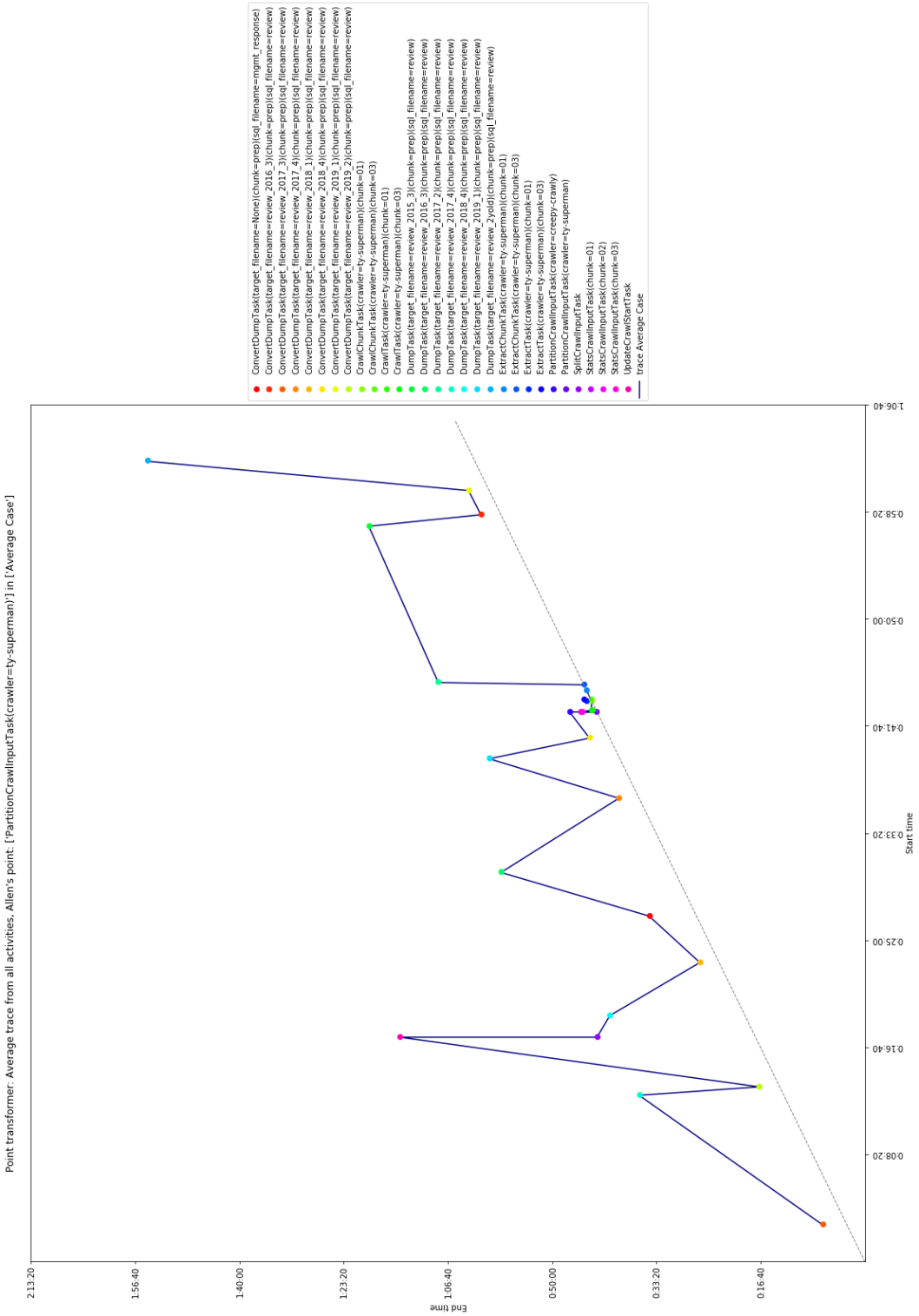


Figure A.6: Extended: figure 3.6(b)

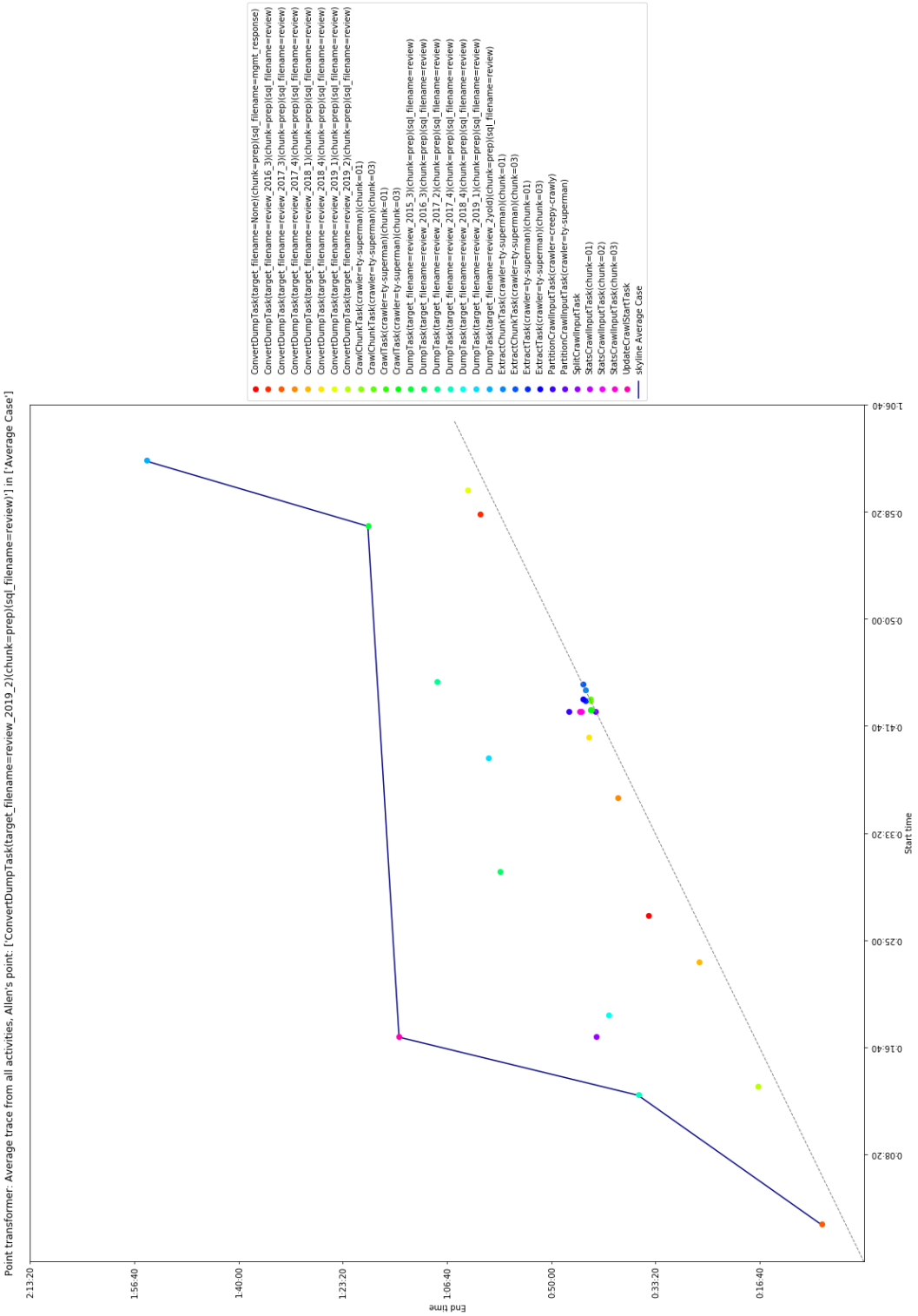


Figure A.7: Extended: figure 3.7(b)

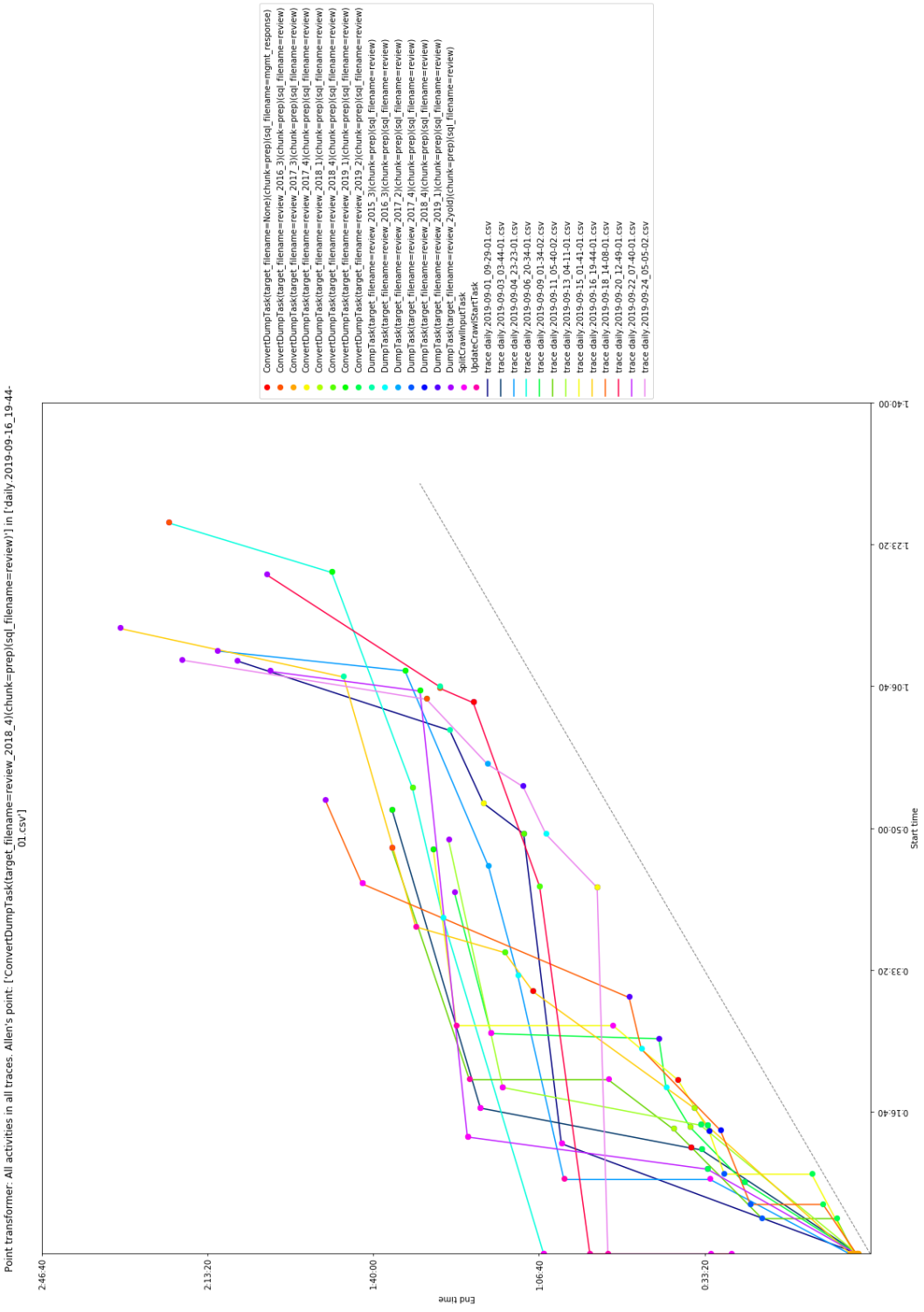


Figure A.8: Extended: figure 3.7(b)

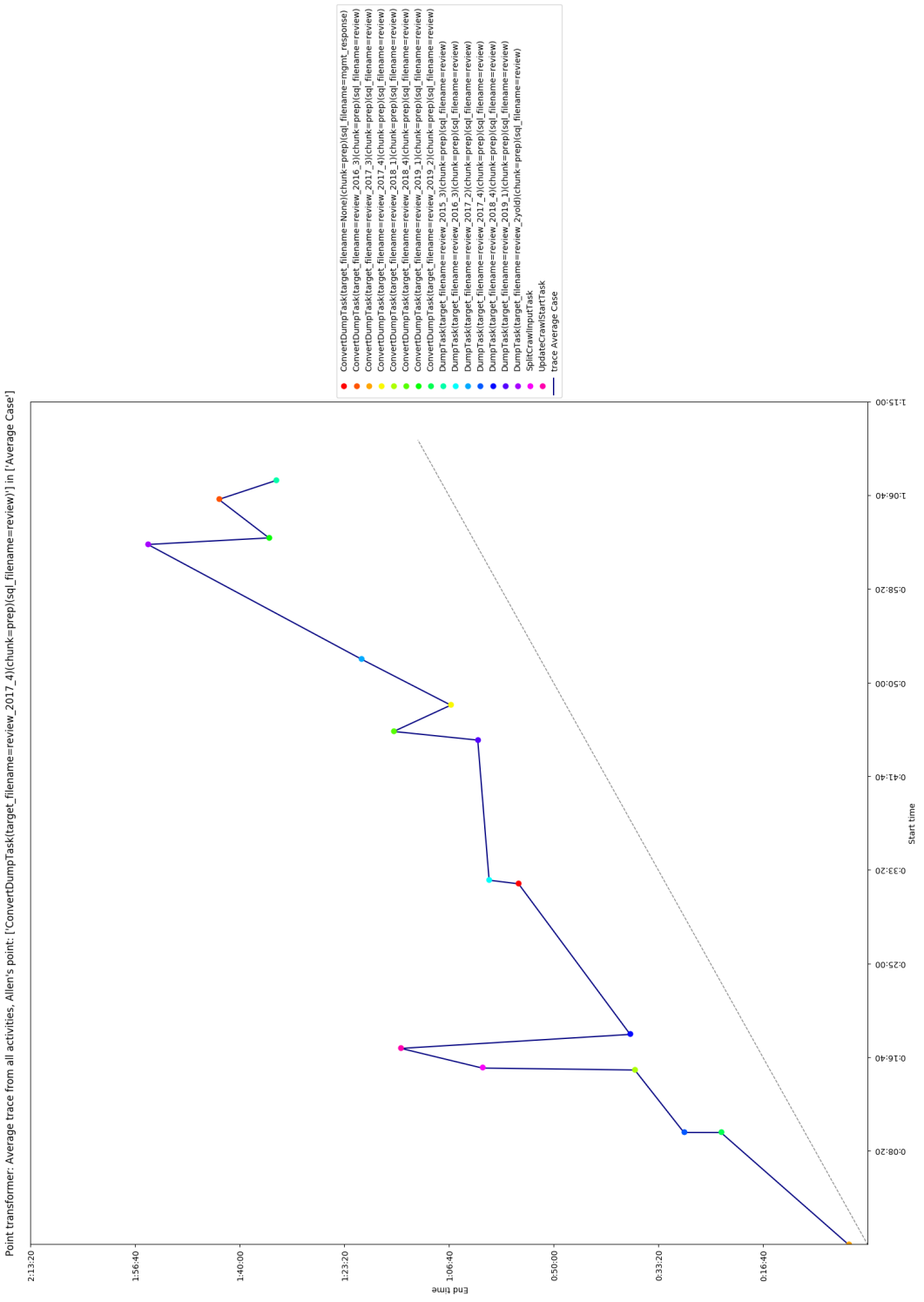


Figure A.9: Extended: figure 3.8(b)

Figure A.10: Number of events in trace anomaly per case

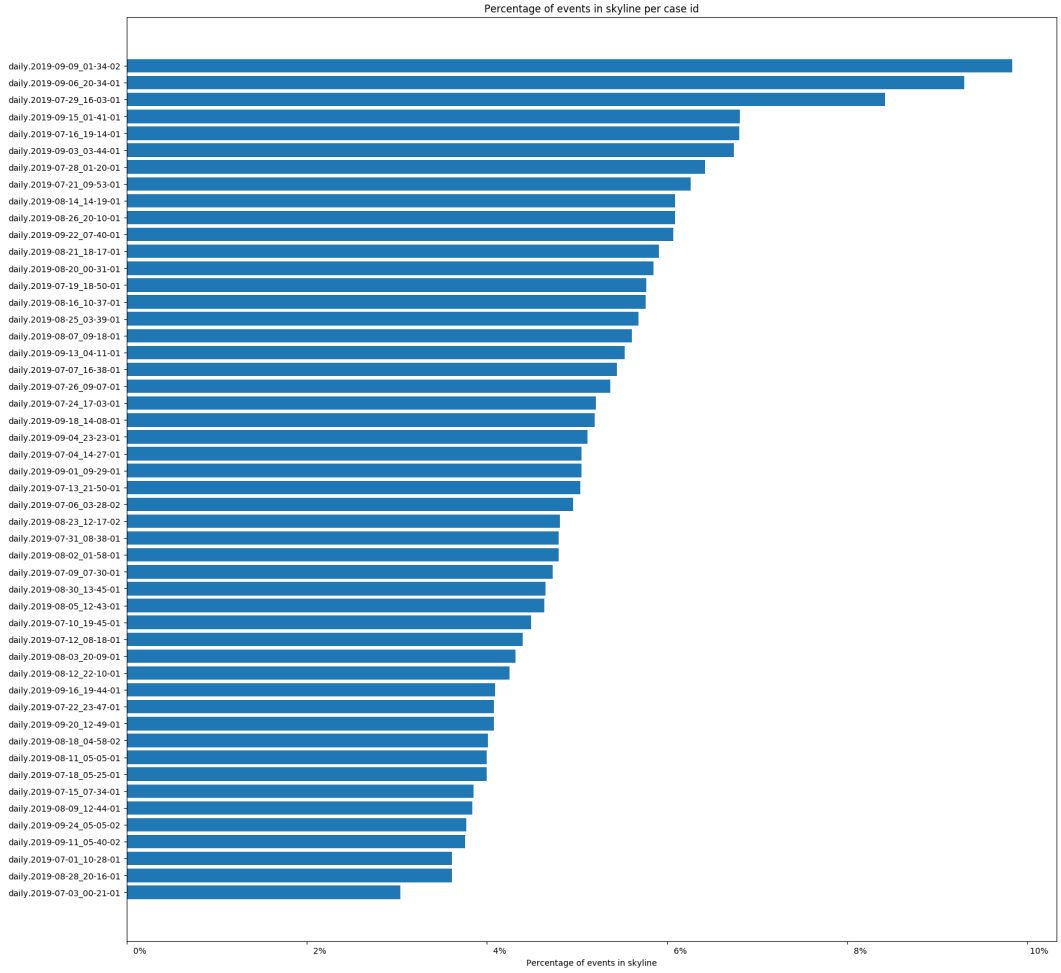


Figure A.11: Percentage of events in skyline anomaly per case

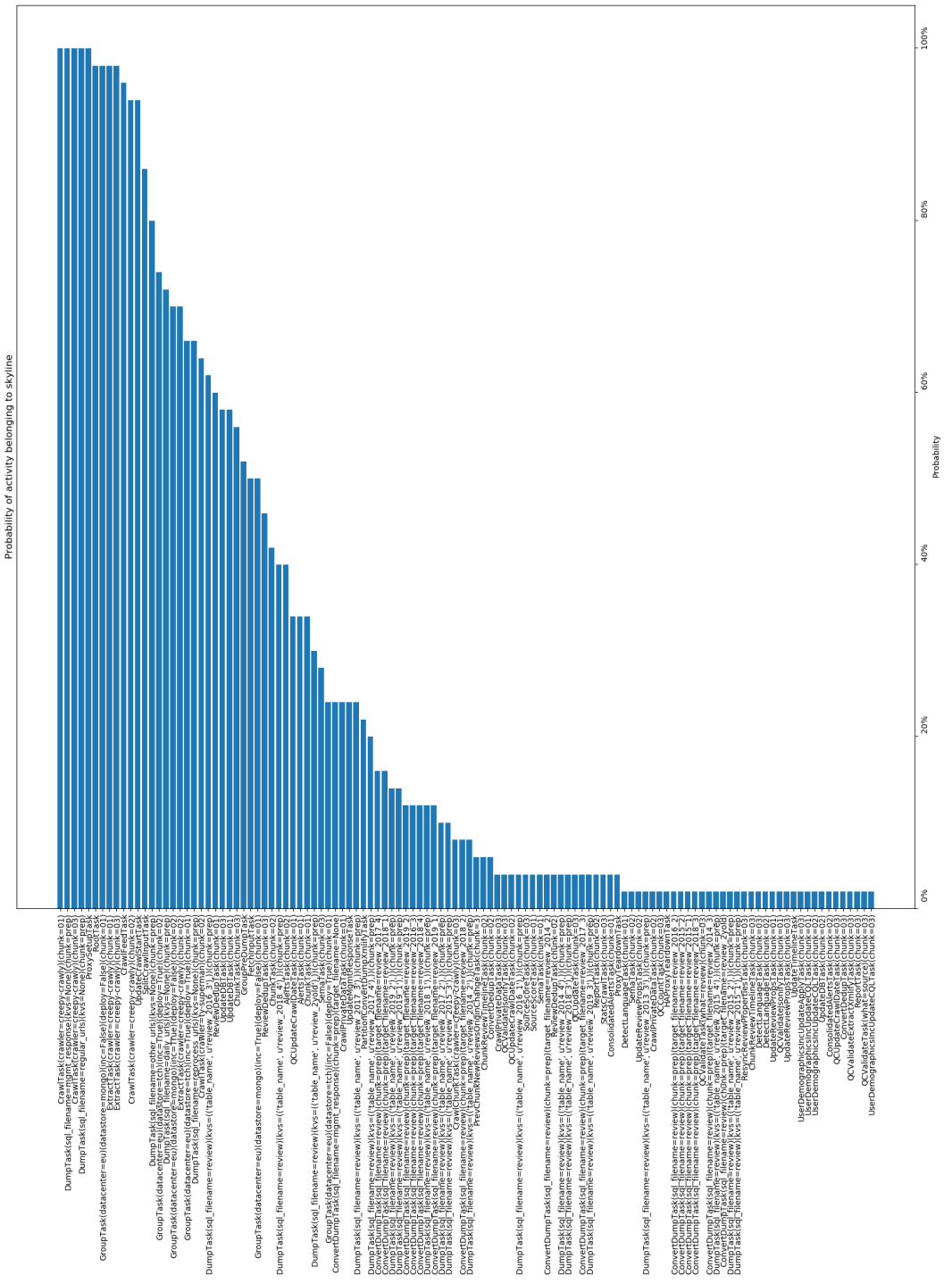


Figure A.12: Activity id by it probability to appear on the skyline for `daily` as in section 4.1

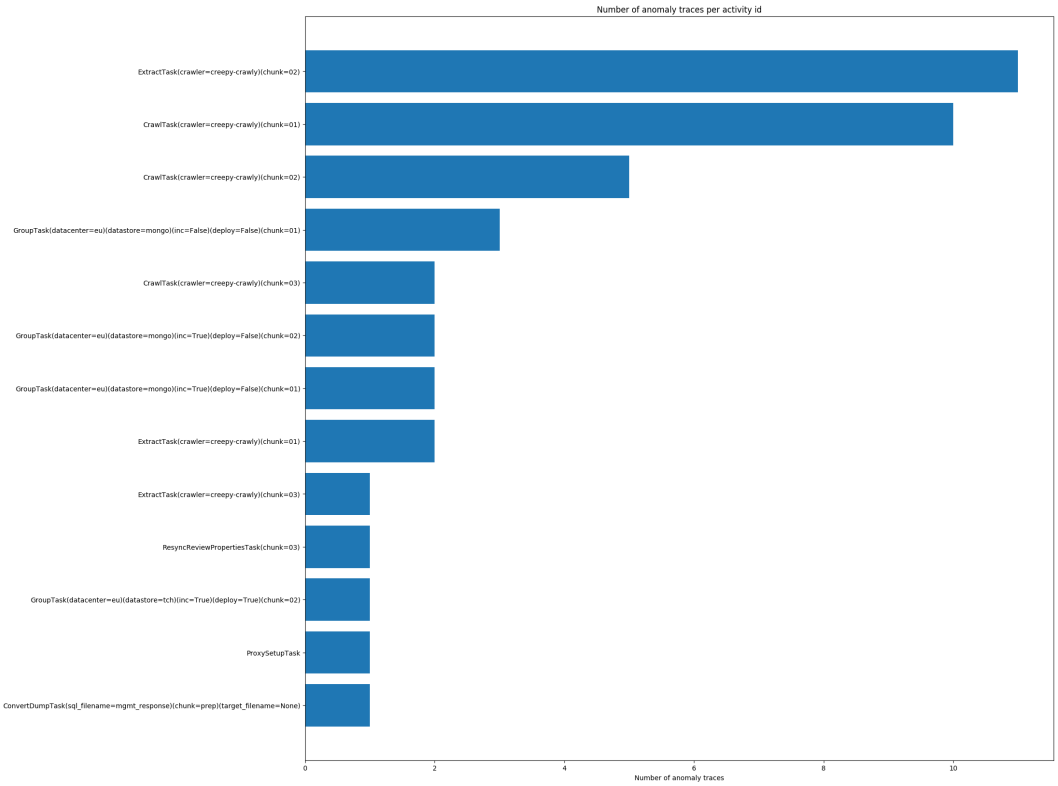


Figure A.13: Number of traces missing activity of interest

List of Figures

List of Tables

Bibliography

[1] Wil Aalst and A.K.A. Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 02 2005.

[2] J.F. Allen. Maintaining Knowledge About Temporal Intervals. volume 26, pages 832–843, New York, NY, USA, 1983. ACM.

[3] C. Atkins. Prescription or description: some observations on the conceptual modelling process. In *Proceedings 1996 International Conference Software Engineering: Education and Practice*, pages 34–41, 1996.

[4] Andrea Burattin. *Introduction*, pages 1–7. Springer International Publishing, Cham, 2015.

[5] Andrea Burattin and Alessandro Sperduti. Heuristics miner for time intervals. volume 207, 01 2015.

[6] Andrea Burattin, Alessandro Sperduti, and Marco Veluscek. Business models enhancement through discovery of roles. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, 16-19 April, 2013*, pages 103–110, 2013.

[7] Vadim Denisov, Elena Belkina, Dirk Fahland, and Wil M.P. van der Aalst. The performance spectrum miner: visual analytics for fine-grained performance analysis of processes. In F. Casati, editor, *BPMTracks 2018*, CEUR Workshop Proceedings, pages 96–100. CEUR-WS.org, 9 2018.

[8] Lucantonio Ghionna, Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Outlier detection techniques for process mining applications. volume 4994, pages 150–159, 05 2008.

[9] T. Guyet and R. Quiniou. Mining temporal patterns with quantitative intervals. In *2008 IEEE International Conference on Data Mining Workshops*, pages 218–227, 2008.

[10] Eva L. Klijn and Dirk Fahland. Performance mining for batch processing using the performance spectrum. In *15th International Workshop on Business Process Intelligence*, 2019.

[11] Jens Wischnewsky Marwan Hassani, Yifeng Lu and Thomas Seidl. A geometric approach for mining sequential patterns in interval-based data streams. *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 2128–2135, 2016.

[12] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[13] Florian Richter and Thomas Seidl. Tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management*, pages 289–305, Cham, 2017. Springer International Publishing.

[14] Pedro Robledo. Process mining plays an essential role in digital transformation. In *https://medium.com/@pedrorobledobpm/process-mining-plays-an-essential-role-in-digital-transformation-384839236bbe*.

[15] Andreas Rogge-Solti and Gjergji Kasneci. Temporal anomaly detection in business processes. In Shazia Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, pages 234–249, Cham, 2014. Springer International Publishing.

[16] Guangchen Ruan and Beth Plale. Parallel and quantitative sequential pattern mining for large-scale interval-based temporal data. pages 32–39, 01 2015.

[17] Jesse Santiago and Desirae Magallon. Critical path method. *CEE 320 – VDC SEMINAR*, feb 2009.

[18] Ioannis Tsamardinos. *Constraint-Based Temporal Reasoning Algorithms with Applications to Planning*. PhD thesis, 01 2001.

[19] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, Heidelberg, 2 edition, 2016.