RWTH AACHEN UNIVERSITY

The present work was submitted to the chair of Process and Data Science

Bachelor Thesis

Localized Conformance and Performance Analysis based on Event Data: Diagnosing Individual Places

Daniel Tacke genannt Unterberg

supervised by
Prof. Wil M.P. van der Aalst
Prof. Martin Grohe

PADS RWTH University
February 27, 2019

Abstract

Nowadays, event data is being generated by many different sources. From process aware information systems (PAIS) to social media and medical devices. Answering questions about conformance and performance of the processes behind the logged data is the goal of all process mining activities.

To fully uncover complex patterns and problems in the process, several perspectives on the data need to be combined. Additionally, since some problems may only occur at certain times or stages in the process, the analysis needs to be fine grained. Most state-of-the-art approaches average over the entire process duration which obscures many weak or rare patterns.

To consider the different perspectives in higher detail, we propose an approach to localize conformance and performance to individual places in a model and time intervals of the process duration.

Contents

Chapter 1

Introduction

Process aware information systems (PAIS) are becoming more and more widespread. These information systems log huge amounts of events, and even more event data can be extracted from all other kinds of ubiquitous organizational databases [15]. Additionally, in our increasingly digitized world, not just business processes are being recorded. Social media, smartphone sensors (e.g. location) and medical devices are just some of the other possible sources of event data. The goal of process mining is to exploit these masses of event data to answer two types of fundamental questions regarding the process behind the data.

1. *Conformance* — in what way does the observed behavior deviate from the norm? Are rules and regulations being followed?

2. *Performance* — how long do certain tasks take? What are the bottlenecks?

A prerequisite to answering these questions is to have a process model to compare the data to. Models which describe the way the process *should* be performed are called *de jure* models. They can be hand-made and exist separately from the observed data. If such a model does not exist, approaches using the mechanism *discovery* are used to create so-called *de facto* models which are solely based on the observed data. Handling noise, incompleteness and Big Data is part of ongoing research for improving this mechanism. Figure 1.1 also shows the other mechanism *replay* which confronts the given log with a given or discovered model. Approaches using this mechanism can create all kinds of diagnostics like metrics, descriptions or enriched models. Enriching a model entails projecting useful information, extracted from the data, on the structure of the model, e.g. marking where deviations take place.

This enables insights into different perspectives on the two fundamental questions. We focus on the notion of *fitness* for conformance and *throughput* for performance. Fitness measures how well the model is able to mimic the observed behavior. This in turn can be used to detect where behavior occurred in reality that was not intended. Throughput ranges from case duration to waiting times between events. The influence of *data* attributes, like who executed an event, and *process context* is considered as well. Process context ranges from local, like how many cases are currently active (busyness), to external like the weather. We only measure the local process context in the form of busyness here since external context is almost impossible to handle.

A shortcoming of state-of-the-art approaches is that they often give overall results and averages. For example, one fitness value for the entire log and model. This is useful for

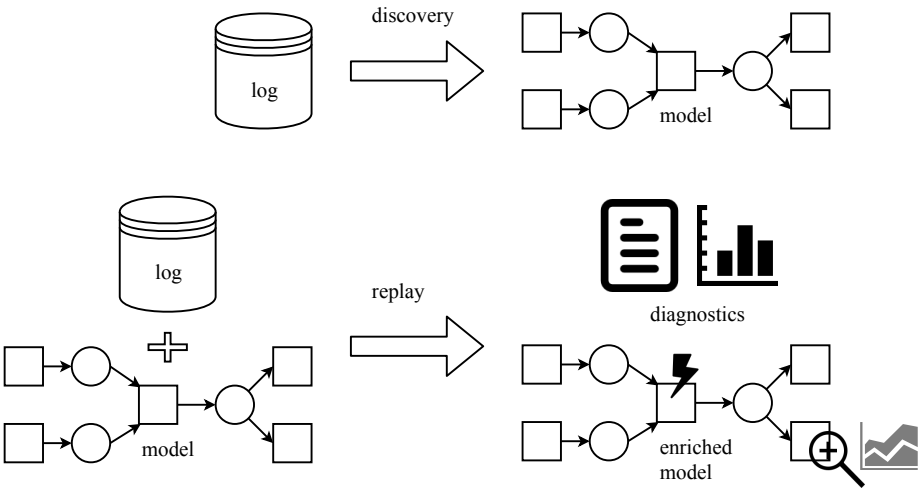


Figure 1.1: Process mining mechanisms

determining if a model has sufficient fitness but not for finding out where the deviations take place. Even more important is the fact that real-life processes are often dynamic and change in some way over time which is called concept drift. That means a low fitness or bad throughput may indicate that the process was always this way or that it behaved perfectly for the first part of the observed timeframe and only got worse after that. Additionally, even the correlations between several measured aspects may be variable over time. A low fitness could lead to bad performance, but this might also reverse.

As an example, consider the Petri net model given in Figure 1.2. It describes the control-flow of a small credit application process where every application starts with a submission. Then, the credit history of the applicant is checked. If the result is *OK*, the application should be accepted and otherwise rejected. After accepting, the credit is provisioned, and the payment is setup concurrently. Finally, the credit is sent. The information system logs the credit amount for cases, time and type of activity for events but no resource information.

However, in real-life, a certain employee is not following protocol. Whenever the load at his position is high and no one notices, he simply accepts low value applications where the credit history check was negative. Additionally, he actively delays high value applications, making them wait for acceptance longer. All of this only happens for a short part of the whole logged time span.

To discover such a complex pattern in full detail in the event data given the model, multiple perspectives are necessary. More specifically,

- *Fitness* — Applications which should be rejected are accepted
- *Throughput* — Applications are being delayed
- *Data* — Only low value applications deviate, and only high value applications are delayed
- *Process Context* (local) — All of this only happens when the affected places are busy, i.e. many cases are currently waiting there

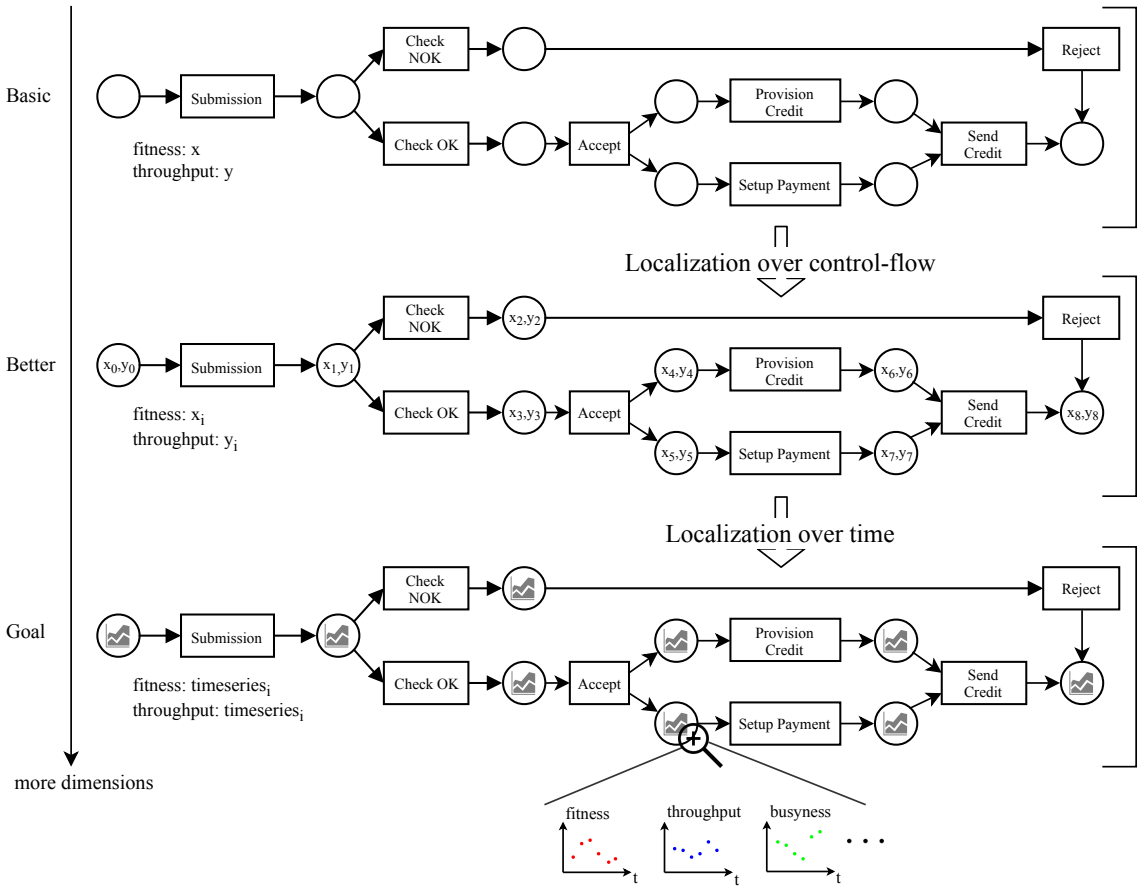


Figure 1.2: Overview of the approach

Condensing the perspectives into single results would obscure the entire pattern. The fitness would still be near perfect, since only specific cases deviate and only during a certain short timeframe. This is also true for the other perspectives. As a first step, as shown in the overview of our approach in Figure 1.2, we localize the perspectives over *control-flow*. That means we can now differentiate the stages of the process. This allows us to detect a stronger deviation on the place after CHECK_NOK and longer throughput times on the place after CHECK_OK. However, averaging over the whole timespan would still not allow us to find out when it happened. The next step is localizing over *time*, leading to the goal of having a timeseries for each place of the model. These finally make it possible to fully discover the problematic pattern described above.

The example shows that averaging, especially over time, can make it impossible to detect complex patterns in event logs. To solve this problem, we propose localizing conformance and performance over control-flow by considering each place individually and then localizing it further over time by discretizing the process duration.

Chapter 2 covers the preliminaries necessary for our work. In Chapter 3, we present our approach. Following that, the implementation in ProM is detailed in Chapter 4. Then, the approach is evaluated on synthetic and real-life event logs in Chapter 5. Second to last, related work is discussed in Chapter 6 and the report is concluded in Chapter 7.

Chapter 2

Preliminaries

This chapter uses definitions adapted from [14, 6, 2] to give a background of the formalization of event logs, Petri nets and alignments. First off, some basic mathematical foundations are required.

Definition 2.1 (Tuples)**.** For sets $X_1, X_2, \ldots, X_n$, $t \in X_1 \times X_2 \times \ldots \times X_n$ is an n-tuple. For $1 \leq i \leq n$, t_i is the projection on the ith element in the tuple.

Definition 2.2 (Functions)**.** For sets X and Y, $f : X \to Y$ is a total function and $g : X \nrightarrow Y$ is a partial function. The domain of a function is the set of elements it is defined on, so $dom(f) = X$ and $dom(g) \subseteq X$ because a partial function does not have to be defined on every element. All elements in the domain are mapped to an element in the target set Y.

Definition 2.3 (Powerset)**.** For a set X, $\mathcal{P}(X) = \{S | S \subseteq X\}$ denotes the set of all subsets of X.

Definition 2.4 (Multiset)**.** For a set X, a multiset $B : X \to \mathbb{N}_0$ over X is a set with duplicates. $B(x)$ gives the frequency of x in B. Multisets use square brackets e.g. $B_1 = [a, a, b, c, c, c] = [a^2, b, c^4], B_2 = [a, d]$. They overload the typical set operations $B_1 \uplus B_2 = [a^3, b, c^4, d], B_1 \cap B_2 = [a]$ and $B_1 \setminus B_2 = [a, b, c^4]$. The set of all multisets over X is $\mathcal{B}(X)$.

Definition 2.5 (Sequences)**.** For any set X, $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle \in X^*$ is a finite sequence over X of length $|\sigma| = n$. $set(\sigma) = \{x \in X | \exists 1 \leq i \leq |\sigma| : \sigma_i = x\}$ is the set of distinct elements in the sequence σ. Two sequences $\sigma, \sigma' \in X^*$ can be concatenated to $\sigma \cdot \sigma'$, e.g. $\langle a, b \rangle \cdot \langle c \rangle = \langle a, b, c \rangle$. Any (partial) function $f : X \nrightarrow Y$ to some set Y can be applied to a sequence over X recursively as follows. $f(\langle\rangle) = \langle\rangle$ and for $x \in X$ and $\sigma \in X^*$:

$$f(\langle x \rangle \cdot \sigma) = \begin{cases} f(\sigma) & \text{if } x \notin dom(f) \\ \langle f(x) \rangle \cdot f(\sigma) & \text{if } x \in dom(f) \end{cases}$$

A sequence over X can also be projected onto a subset $Q \subseteq X$ recursively. So $\langle\rangle \restriction_Q = \langle\rangle$ and for $x \in X$ and $\sigma \in X^*$:

$$(\langle x \rangle \cdot \sigma) \restriction_Q = \begin{cases} \sigma \restriction_Q & \text{if } x \notin Q \\ \langle x \rangle \cdot \sigma \restriction_Q & \text{if } x \in Q \end{cases}$$

Sequences over tuples $\sigma \in (X_1 \times \ldots \times X_m)^*$ can be projected to a sequence over X_i $(1 \leq i \leq m)$ with $\sigma \restriction_i$.

Definition 2.6 (Logic)**.** For a set X and a formula $\phi : X \to \mathcal{B}$, $\forall x \in X : \phi(x)$ requires ϕ to hold for all x, $\exists x \in X : \phi(x)$ requires ϕ to hold for at least one x and $\exists!\ x \in X : \phi(x)$ requires ϕ to hold for exactly one x. For two formulas ϕ and ψ, $\phi \wedge \psi$ requires both to hold, $\phi \vee \psi$ requires at least one to hold and $\phi\ XOR\ \psi$ requires exactly one of them to hold.

The recording of events during process execution is the basis of all process mining activities. Recorded events are always associated with an activity and a time. Other attributes are optional.

Definition 2.7 (Event)**.** Let $\mathcal{E}$ denote the universe of all possible events. The set ATE contains all possible event attribute names. For an event $e \in \mathcal{E}$ and attribute name $n \in ATE$, $n(e)$ is the value of attribute n of event e. If e does not have attribute n, $n(e) = \perp$. $\mathcal{U}_A$ denotes the universe of all activities. The event attributes $activity : \mathcal{E} \to \mathcal{U}_A$ and $time : \mathcal{E} \to Time$ are always defined.

For readability, we write events together with their activity in the superscript like $e^{activity(e)}$. Traces are a chronologically ordered sequence of such events and belong to a case along with possible further case attributes. Since events are considered to be unique, there cannot be any duplicates in a trace.

Definition 2.8 (Trace)**.** A trace $\sigma = \langle e_1, e_2, \ldots, e_n \rangle \in \mathcal{E}^*$ is a finite sequence of events without duplicates, i.e. $\forall 1 \leq i < j \leq n : e_i \neq e_j$. The events are non-strictly ordered by their time attribute, i.e. $\forall 1 \leq i < j \leq n : time(e_i) \leq time(e_j)$.

Like an event, a case is a unique object with attributes, one of which being the associated trace.

Definition 2.9 (Case)**.** Let $\mathcal{C}$ denote the universe of all possible cases. The set ATT contains all possible case attribute names. For a case $c \in \mathcal{C}$ and attribute name $n \in ATT$, $n(c)$ is the value of attribute n. If c does not have attribute n, $n(c) = \perp$. The attribute $trace(c) = \hat{c} \in \mathcal{E}^*$ is always defined. $\hat{c}$ will be used as a shorthand for the associated trace of a case.

An event log is a collection of cases.

Definition 2.10 (Event log)**.** Let $\mathcal{L} \subseteq \mathcal{C}$. $\mathcal{L}$ is an event log if every event in the log is unique, i.e. $\forall c, c' \in \mathcal{L} : c \neq c' \implies set(\hat{c}) \cap set(\hat{c}') = \emptyset$. The universe of event logs is $\mathcal{U}_L$.

As a notation for process models, we use (labeled) Petri nets. Observe that, since different process notations (like BPMN models) can be converted between each other, this is not a restriction.

Definition 2.11 (Labeled Petri net)**.** A (labeled) Petri net is a tuple $PN = (P, T, F, l)$ consisting of the set of places P, set of transitions T with $P \cap T = \emptyset$, a flow relation $F \subseteq (P \times T) \cup (T \times P)$ and a labeling function $l : T \to \mathcal{U}_A \cup \{\tau\}$, with $\tau \notin \mathcal{U}_A$. The labeling function maps each visible transition to an activity in $\mathcal{U}_A$. Transitions $t \in T$ mapped to τ are called silent or invisible. They are unobservable. The sets $T^l = \{t \in T | l(t) \neq \tau\}$ and $T^\tau = \{t \in T | l(t) = \tau\}$ partition the transitions into those labeled with an activity and the silent ones.

To be able to refer to adjacent nodes more easily, the preset and postset are defined. For places they contain transitions and for transitions places.

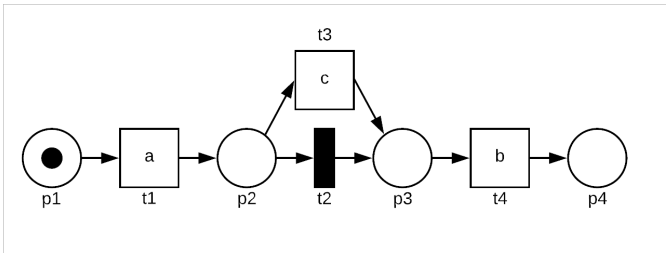


Figure 2.1: A simple Petri net

$$\gamma_1 = \begin{array}{|c|c|c|c|} \hline \begin{matrix} e_{11} \\ \text{b} \end{matrix} & \begin{matrix} e_{12} \\ \text{a} \end{matrix} & \begin{matrix} e_{13} \\ \text{c} \end{matrix} & \gg \\ \hline \gg & \begin{matrix} \text{a} \\ \text{t1} \end{matrix} & \begin{matrix} \text{c} \\ \text{t3} \end{matrix} & \begin{matrix} \text{b} \\ \text{t4} \end{matrix} \\ \hline \end{array} \text{ and } \gamma_2 = \begin{array}{|c|c|c|} \hline \begin{matrix} e_{21} \\ \text{a} \end{matrix} & \gg & \begin{matrix} e_{22} \\ \text{b} \end{matrix} \\ \hline \begin{matrix} \text{a} \\ \text{t1} \end{matrix} & \begin{matrix} \tau \\ \text{t2} \end{matrix} & \begin{matrix} \text{b} \\ \text{t4} \end{matrix} \\ \hline \end{array}$$

Figure 2.2: Two example alignments for $\hat{c}_1 = \langle e^b_{11}, e^a_{12}, e^c_{13} \rangle$ and $\hat{c}_2 = \langle e^a_{21}, e^b_{22} \rangle$

Definition 2.12 (Preset, postset)**.** Let $PN = (P, T, F, l)$ be a Petri net. For every node $n \in P \cup T$, $n^\bullet = \{m | (n, m) \in F\}$ is the postset and ${}^\bullet n = \{m | (m, n) \in F\}$ the preset of n.

A marking describes the state of a Petri net, represented by a distribution of tokens on the places.

Definition 2.13 (Marking)**.** Let $PN = (P, T, F, l)$ be a Petri net. A marking $M \in \mathcal{B}(P)$ is a multiset of places. For every $p \in P$, $M(p)$ is the number of tokens on place p. A transition $t \in T$ is enabled in a marking M, written $M[t\rangle$, if each place in ${}^\bullet t$ has at least one token. The transition execution $M[t\rangle M'$ results in the marking $M' = (M \setminus {}^\bullet t) \uplus t^\bullet$ where one token is removed from every place in the preset and one is added to every place in the postset.

To make a Petri net executable, an initial marking which describes the initial distribution of tokens and a final marking which describes a valid completion states are required.

Definition 2.14 (System net)**.** A system net $SN = (PN, M_i, M_f)$ is a Petri net $PN = (P, T, F, l)$ with an initial marking M_i and final marking M_f. A valid firing sequence on SN is a sequence of transitions $\sigma \in T^*$ such that $\forall 1 \leq j \leq |\sigma| : M_{j-1}[\sigma_j\rangle M_j$ with $M_0 = M_i$ and $M_{|\sigma|} = M_f$ where every transition execution was enabled.

For replaying a (possibly unfitting) trace on a system net which might contain silent transitions or multiple transitions with the same label (duplicate transitions) alignments from [18] are being used.

Definition 2.15 (Alignment)**.** Let $SN = (PN, M_i, M_f)$ be a system net and $c \in \mathcal{C}$ a case. An alignment $\gamma \in (\mathcal{E} \cup \{\gg\}) \times (T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ is a sequence of so-called moves such that $\gamma{\restriction_1}{\restriction_\mathcal{E}} = \hat{c}$ and $\gamma{\restriction_2}{\restriction_T}$ is a valid firing sequence on SN. And for every $1 \leq i \leq |\gamma|$:

- $\gamma_i = (e, t)$ with $e \in \mathcal{E}$, $t \in T$ is called a move in both and if $activity(e) = l(t)$ it is called a synchronous move,
- $\gamma_i = (e, \gg)$ with $e \in \mathcal{E}$ is called a log move,

- $\gamma_i = (\gg, t)$ with $t \in T$ is called a model move and if $l(t) = \tau$ it is called an invisible move.

Since there may be multiple valid alignments, a cost function is used to rank them and return the optimal one. The standard cost function, which is often used, gives sync moves and invisible moves cost 0 and move on model/log cost 1. A move on both which is not a synchronous move gets a cost of ∞ since it is usually not wanted. For the cost of the whole alignment, the costs for all moves are summed up.

Figure 2.2 shows two example alignments for the traces $\hat{c}_1 = \langle e^b_{11}, e^a_{12}, e^c_{13} \rangle$ and $\hat{c}_2 = \langle e^a_{21}, e^b_{22} \rangle$ on the model given in Figure 2.1. The top row contains events together with their *activity* and the bottom row transitions together with their label to make it more readable. The alignments provide a path through the model even for unfitting traces like $\hat{c}_1$ or traces which require the execution of invisible transitions like $\hat{c}_2$.

Chapter 3

Approach

Analyzing conformance and performance expressed as averages over the whole model and log is very broad. Many real-life processes contain complex patterns and behavior as shown in the introduction. They can only be discovered when combining multiple perspectives on the data. The approach presented here focuses on localizing metrics for conformance, performance and process context to individual places in the Petri net model. The metrics are further localized to time intervals to make changes over time visible.

The approach consists of three steps:

1. The log is projected onto the places of the process model to extract a *locally mapped log*
2. *Interactions* are extracted from this *locally mapped log*
3. Metrics are calculated from *interactions* for time intervals

In the following, we introduce these steps in order.

3.1 Locally mapping the log

The first step requires some pre-processing. To properly handle the start and end places in the next steps, we insert unique start and end events into every trace. Their *activity* is $\triangleright$ or $\square$ respectively. Their *time* attribute is the time of first or last event respectively. Let $\mathcal{E}' \supset \mathcal{E}$ be the universe of events including all artificial start and end marker events. To add the corresponding $\triangleright$ and $\square$ transitions in the model, a system net $SN = (PN, M_i, M_f)$ with initial and final marking is required. The $\triangleright$ transition $t_\triangleright$ is connected to all places in M_i and the $\square$ transition $t_\square$ from all places in M_f. So for $PN = (P, T, F, l)$ the pre-processed Petri net is $PN' = (P, T', F', l')$ with $F' = F \cup \{(t_\triangleright, p) | p \in M_i\} \cup \{(p, t_\square) | p \in M_f\}$, $T' = T \cup \{t_\triangleright, t_\square\}$, $\forall t \in T : l'(t) = l(t)$ and $l'(t_\triangleright) = \triangleright, l'(t_\square)) = \square$. Figure 3.1 shows the pre-processed version of Figure 2.1.

From this point on, we assume all Petri nets and logs to be pre-processed in this way. We partition the adjacent transitions of a place into visible and invisible transitions, so for a place $p \in P$, $adj(p) = ({}^\bullet p \cup p^\bullet) \cap T^l$ is the set of adjacent visible transitions and $adj_\tau(p) = ({}^\bullet p \cup p^\bullet) \cap T^\tau$ is the set of adjacent silent transitions. To be able to add silent transition executions, let $l_\tau : T^\tau \to A_\tau$ be a labeling function which assigns a unique artificial activity to every otherwise unlabeled τ-transition. To further define the silent execution events, let $\mathcal{I}$ be the universe of all artificial silent (hence not in $\mathcal{E}'$) events whose

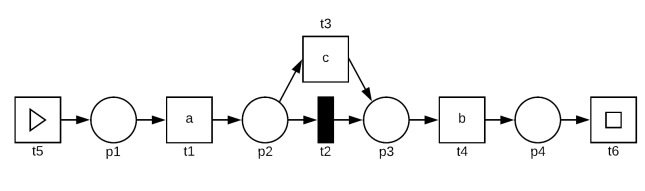


Figure 3.1: The simple model after pre-processing

activity attribute is a silent activity. With these definitions, a locally mapped case can be defined. It is a mapping of events from a case to transitions adjacent to a particular place.

Definition 3.1 (Locally mapped case)**.** Let $\mathcal{L} \in \mathcal{U}_L$ be a log and $PN = (P, T, F, l)$ be a Petri net. For a case $c \in \mathcal{L}$ and place $p \in P$, a locally mapped case lmc for p inherits (and overloads) all attributes of c except for the *trace* attribute. Its trace is overloaded to a sequence of pairs of real events from $\hat{c}$ and inserted artificial silent events together with a fitting transition, i.e. $\hat{lmc} = trace(lmc) \in ((\mathcal{E}' \times adj(p)) \cup (\mathcal{I} \times adj_\tau(p)))^*$. For every pair $(e, t) \in \mathcal{E}' \times adj(p)$ or $(i, t') \in \mathcal{I} \times adj_\tau(p)$ in the sequence, the label of the transition has to match the activity of the event, i.e. $activity(e) = l(t)$ and $activity(i) = l_\tau(t')$. All events in the sequence have to be unique and ordered by *time*, so $\forall 1 \leq i < j \leq |\hat{lmc}|, \hat{lmc}_i = (e, t), \hat{lmc}_j = (e', t') : e \neq e' \wedge time(e) \leq time(e')$. Lastly, the projection onto real events is a subset of the original trace, i.e. $set(\hat{lmc}{\restriction_1}{\restriction_{\mathcal{E}'}}) \subseteq set(\hat{c})$. If $|\hat{lmc}| = 0$, lmc is called an empty locally mapped case which means the case did not interact with the place p.

There are many ways to map a case onto the locally mapped cases, so we first define a generalized strategy. The strategy also determines the mandatory *time* attribute of inserted silent events.

Definition 3.2 (Case mapping strategy)**.** Let $\mathcal{L} \in \mathcal{U}_L$ be a log and $PN = (P, T, F, l)$ a Petri net. For any case $c \in \mathcal{L}$, a case mapping strategy $case\_map\_strat(c) = (lmc_p)_{p \in P}$ creates a locally mapped case for each place $p \in P$. It needs to be consistent because a transition execution mapped in one of the locally mapped cases needs to be included in all the others also adjacent to the transition. So, $\forall p \in P\ \forall (e, t) \in set(\hat{lmc}_p)\ \forall p' \in ({}^\bullet t \cup t^\bullet) : (e, t) \in set(\hat{lmc}_{p'})$.

Consistency can be achieved by first selecting a subset of events, mapping them to transitions, adding silent executions and then projecting them onto the adjacent transitions of the places. Now, the entire event log can be mapped onto individual places with locally mapped logs.

Definition 3.3 (Locally mapped log)**.** Let $\mathcal{L} \in \mathcal{U}_L$ be a log, $PN = (P, T, F, l)$ be a Petri net and *map_strat* be a case mapping strategy. Then a locally mapped log $\mathcal{L}_p$ of place $p \in P$ is the set of all non-empty locally mapped cases, i.e. $\mathcal{L}_p = \{lmc | lmc = map\_strat(c)_p \wedge |\hat{lmc}| \neq 0 \wedge c \in \mathcal{L}\}$.

We propose a specific case mapping strategy *sync_strat* where a global execution is mapped locally. It uses alignments to be able to handle duplicate transitions and silent transition executions. The synchronous moves already provide a mapping of a subset of events from the trace to transitions in the model, so only the silent transition executions

Algorithm 1 The proposed *map_strat*

Require: given $PN = (P, T, F, l)$
  $\forall p \in P : lmc_p \leftarrow$ new locally mapped case
  $\forall p \in P : \hat{lmc}_p \leftarrow \langle\rangle$
  $M \leftarrow$ new empty marking
  $lastMappedTime : P \nrightarrow Time$
  function MAPSTRAT(alignment γ, boolean *mapLogMoves*)
    for $i = 1 : |\gamma|$ **do**
      if $\gamma_i = (e, t) \wedge activity(a) = l(t)$ **then** ▷ sync move
        HANDLETRANSITION(e, t)
      else if $\gamma_i = (e, \gg)$ **then** ▷ log move
        if $mapLogMoves \wedge \exists t \in T : l(t) = activity(e)$ **then**
          HANDLETRANSITION(e, t)
      else if $\gamma_i = (\gg, t) \wedge l(t) = \tau$ **then** ▷ inv move
        if $M[t\rangle$ **then**
          $enabledAt \leftarrow \max\{lastMappedTime(p) | p \in {}^\bullet t\}$
          $e_\tau \leftarrow$ new unique event
          $activity(e_\tau) \leftarrow l_\tau(t)$
          $time(e_\tau) \leftarrow enabledAt$
          HANDLETRANSITION(e_τ, t)
    return $(lmc_p)_{p \in P}$
  function HANDLETRANSITION(event e, transition t)
    $M \leftarrow (M \setminus {}^\bullet t) \uplus t^\bullet$
    for all $p \in {}^\bullet t \cup t^\bullet$ **do**
      $\hat{lmc}_p \leftarrow \hat{lmc}_p \cdot (e, t)$
      if $p \in t^\bullet$ **then**
        $lastMappedTime(p) \leftarrow time(e)$

have to be added. This is done by replaying the synchronous and invisible moves on the model in a token-based manner. Synchronous moves are always executed, even if not enabled. Invisible moves are only executed if they are enabled, to be able to set their *time* attribute which is the instant they are enabled. That means chains of silent events consistently shift the sojourn duration between two real events to the place at the end of the chain. The invisible move chains can be cut-off by non-invisible model moves, leaving the deviation of missing this event on the places adjacent to these non-invisible model moves. This way, the locally mapped cases can be seen as a projection of the fitting parts of the event log onto locally mapped logs for each place. A variant *all_strat* also tries to consider log moves which makes it unsuitable for models with duplicate transitions. It essentially treats log moves which can be mapped to a transition as synchronous moves, executing them unconditionally. More information from the log is used that way. Especially loops in a trace which cannot be mimicked by the model and hence end up being log moves can contribute to the analysis this way.

A configurable algorithm for both variants is presented in Algorithm 1. Provided an alignment γ of a case, it steps through the sequence of moves and updates the locally mapped cases accordingly in the helper function HANDLETRANSITION. It uses the partial function *lastMappedTime* to track timestamps and a marking M as an updated current

Local log	*sync_strat*
$\mathcal{L}_{p1}$	$\{\langle(e^{\triangleright}_{11}, t5), (e^{a}_{13}, t1)\rangle, \langle(e^{\triangleright}_{21}, t5), (e^{a}_{22}, t1)\rangle\}$
$\mathcal{L}_{p2}$	$\{\langle(e^{a}_{13}, t1), (e^{c}_{14}, t3)\rangle, \langle(e^{a}_{22}, t1), (e^{t2}_{\tau}, t2)\rangle\}$
$\mathcal{L}_{p3}$	$\{\langle(e^{c}_{14}, t3)\rangle, \langle(e^{t2}_{\tau}, t2), (e^{b}_{23}, t4)\rangle\}$
$\mathcal{L}_{p4}$	$\{\langle(e^{\square}_{15}, t6)\rangle, \langle(e^{b}_{23}, t4), (e^{\square}_{24}, t6)\rangle\}$

Local log	*all_strat*
$\mathcal{L}_{p1}$	$\{\langle(e^{\triangleright}_{11}, t5), (e^{a}_{13}, t1)\rangle, \langle(e^{\triangleright}_{21}, t5), (e_{22}, a)\rangle\}$
$\mathcal{L}_{p2}$	$\{\langle(e^{a}_{13}, t1), (e^{c}_{14}, t3)\rangle, \langle(e^{a}_{22}, t1), (e^{t2}_{\tau}, t2)\rangle\}$
$\mathcal{L}_{p3}$	$\{\langle(e^{b}_{12}, t4), (e^{c}_{14}, t3)\rangle, \langle(e^{t2}_{\tau}, t2), (e^{b}_{23}, t4)\rangle\}$
$\mathcal{L}_{p4}$	$\{\langle(e^{b}_{12}, t4), (e^{\square}_{15}, t6)\rangle, \langle(e^{b}_{23}, t4), (e^{\square}_{24}, t6)\rangle\}$

Figure 3.2: Comparison of local log mapping strategies

marking. The transitions of synchronous moves and, if desired, mapped log moves are fired regardless if they are enabled. Invisible model moves are executed if they were actually enabled otherwise they are ignored. When they are executed, a unique new event is created and its mandatory attributes set. Then it is treated like any other transition execution.

As an example, consider the pre-processed log cases $\hat{c_1} = \langle e^{\triangleright}_{11}, e^{b}_{12}, e^{a}_{13}, e^{c}_{14}, e^{\square}_{15}\rangle$ and $\hat{c_2} = \langle e^{\triangleright}_{21}, e^{a}_{22}, e^{b}_{23}, e^{\square}_{24}\rangle\}$ in log $\mathcal{L} = \{c_1, c_2\}$ on the model given in Figure 3.1. The results of both strategies on this log are shown in Figure 3.2. With *sync_strat*, the unfitting event e^{b}_{12} of the first case is not mapped which assigns that deviation to places $p3$ and $p4$. *all_strat* maps this event, which hides the problem from $p4$ and focuses it on $p3$ as swapped events.

3.2 Extracting Interactions

The next step relies on the locally mapped logs to pair up input events with output events. During token-based replay, the execution of an input activity of a place will produce a token on it and the execution of an output activity of the place will eventually consume the token. This is a complete interaction. The time difference between token production and consumption on the place is called sojourn time. If the trace is not fitting perfectly, this results in places where a token will be missing or remaining. These cases are classified as incomplete interactions.

Definition 3.4 (Complete interaction, incomplete interaction)**.** Let $PN = (P, T, F, l)$ be a Petri net and $\hat{lmc} = \langle \hat{lmc}_1, \hat{lmc}_2, \ldots, \hat{lmc}_n\rangle$ be the trace of a locally mapped case of $p \in P$. A complete interaction is a tuple $ci = ((e, t), (e', t'))$ where $(e, t) = \hat{lmc}_i, (e', t') = \hat{lmc}_j$ such that $1 \leq i < j \leq n$ and $t \in {}^\bullet p$ and $t' \in p^\bullet$. All interactions have a *start*, *end* and *duration* attribute. $start(ci) = time(e)$, $end(ci) = time(e')$ and $duration(ci) = end(ci) - start(ci)$. An incomplete interaction ii is a tuple where the missing pair is replaced by $+$ or $-$. So with $(e, t) \in set(\hat{lmc})$: $ii = (+, (e, t))$ for $t \in p^\bullet$, $ii = ((e, t), -)$ for $t \in {}^\bullet p$ and any of those for $t \in {}^\bullet p \cap p^\bullet$. That means self loops can appear in two different incomplete interactions. The attributes are $start(ii) = end(ii) = time(e)$ and $duration(ii) = 0$. Additionally, interactions also further inherit (and overload) the case attributes of the locally mapped case.

To extract these interactions in a sensible way, we define a generic interaction extraction strategy. This heuristic further maps the locally mapped cases to all contained interactions.

Definition 3.5 (Interaction sets, interaction extraction strategy)**.** Let $PN = (P, T, F, l)$ be a Petri net and lmc a locally mapped case of $p \in P$. *CIS* is called a complete interaction set and *IIS* an incomplete interaction set. An interaction extraction strategy $extract\_strat_p(lmc) = (CIS, IIS)$ for place p has to partition all pairs $(e, t) \in set(\hat{lmc})$ into valid complete and incomplete interactions. Pairs with a self loop transition have to appear in two interactions because they first consume a token then produce one again. So, for all $1 \leq j \leq |\hat{lmc}| = n$ with $\hat{lmc}_j = (e, t)$ the following has to hold:

$$t \in {}^\bullet p \implies (\exists!\ j < k \leq n : (\hat{lmc}_j, \hat{lmc}_k) \in CIS)\ XOR\ ((\hat{lmc}_j, -) \in IIS) \tag{3.1}$$

$$t \in p^\bullet \implies (\exists!\ 1 \leq i < j : (\hat{lmc}_i, \hat{lmc}_j) \in CIS)\ XOR\ ((+, \hat{lmc}_j) \in IIS) \tag{3.2}$$

In other words, a self loop either is in two complete interactions, first as an output transition second as an input transition. Or just in one of those complete interactions and in one incomplete one, or both the input and the output transition part of incomplete interactions. We propose a basic algorithm for an *extract_strat* with Algorithm 2. It performs a local token-based replay on the place where events which try to consume non-existent tokens and events which produce tokens that are never consumed are classified as incomplete interactions. It is also configurable by choosing the internal data structure which stores the producing events as either a stack or a queue. This determines the heuristic used to determine which overlapping events belong together. A small example illustrating the difference is given in Figure 3.3. Complete Interactions are double headed arrows and incomplete ones are $\times$. The queue matches the first producer with the first consumer while the stack matches the last producer with the first consumer.

Algorithm 2 The proposed *extract_strat*

Require: given $PN = (P, T, F)$ and place $p \in P$

```
function EXTRACTINTERACTIONS(locally mapped case lmc)
    CIS ← {}
    IIS ← {}
    upTimes ← new(stack)                                  ▷ or new(queue)
    for i = 1 : |lmc^| do
        (e, t) ← lmc^_i
        if t ∈ p• then
            if empty(upTimes) then
                IIS ← IIS ∪ (+, (e, t))
            else
                (e', t') ← remove(upTimes)
                CIS ← CIS ∪ ((e', t'), (e, t))
        if t ∈ •p then
            add(upTimes, (e, t))
    if not empty(upTimes) then
        for all (e, t) ∈ upTimes do
            IIS ← IIS ∪ ((e, t), −)
    return (CIS, IIS)
```

Finally, the interaction sets for the local logs in Figure 3.2 of the running example are given in Figure 3.4. As mentioned before, *sync_strat* spreads out the incomplete

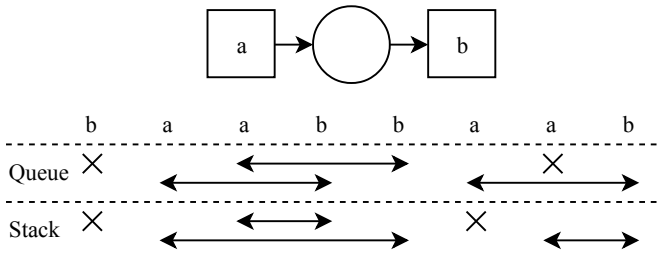


Figure 3.3: A comparison of the queue and stack interaction extraction

Local log	*sync_strat*	
	CIS	*IIS*
$\mathcal{L}_{p1}$	$\{((e^{\triangleright}_{11}, t5), (e^{a}_{13}, t1)), ((e^{\triangleright}_{21}, t5), (e^{a}_{22}, t1))\}$	$\{\}$
$\mathcal{L}_{p2}$	$\{((e^{a}_{13}, t1), (e^{c}_{14}, t3)), ((e^{a}_{22}, t1), (e^{t2}_{\tau}, t2))\}$	$\{\}$
$\mathcal{L}_{p3}$	$\{((e^{t2}_{\tau}, t2), (e^{b}_{23}, t4))\}$	$\{((e^{c}_{14}, t3), -)\}$
$\mathcal{L}_{p4}$	$\{((e^{b}_{23}, t4), (e^{\square}_{24}, t6))\}$	$\{(+, (e^{\square}_{15}, t6))\}$
Local log	*all_strat*	
	CIS	*IIS*
$\mathcal{L}_{p1}$	$\{((e^{\triangleright}_{11}, t5), (e^{a}_{13}, t1)), ((e^{\triangleright}_{21}, t5), (e^{a}_{22}, t1))\}$	$\{\}$
$\mathcal{L}_{p2}$	$\{((e^{a}_{13}, t1), (e^{c}_{14}, t3)), ((e^{a}_{22}, t1), (e^{t2}_{\tau}, t2))\}$	$\{\}$
$\mathcal{L}_{p3}$	$\{((e^{t2}_{\tau}, t2), (e^{b}_{23}, t4))\}$	$\{(+, (e^{b}_{23}, t4)), ((e^{c}_{14}, t3), -)\}$
$\mathcal{L}_{p4}$	$\{((e^{b}_{12}, t4), (e^{\square}_{15}, t6)), ((e^{b}_{23}, t4), (e^{\square}_{24}, t6))\}$	$\{\}$

Figure 3.4: Continued running example with interaction sets

interactions to all adjacent places while *all_strat* focuses it on the input places. Since swapped events can lead to two incomplete interactions, mapping log moves may lead to more incomplete interactions. In this example, using a stack or queue makes no difference.

3.3 Time intervals

Until now, we extracted interactions of the log with places. This only uses the control-flow dimension and ordering of events. Since interactions also have the attributes *start*, *end* and *duration*, it is possible to order them and spread them out over the time dimension. Complete interactions may also have a non-zero duration, so it is not trivial to filter them into time intervals. There are different types of overlap of an interaction with a time interval.

Definition 3.6 (Time interval projections)**.** A time interval $[I_{start}, I_{end})$ has an inclusive start time I_{start} and an exclusive end time I_{end}. The length of the interval is $L = I_{end} - I_{start}$. The projections of an interaction set *IS* (*CIS* or *IIS*) to such a time interval are defined as functions with time interval parameters.

- *Touching*: $T[I_{start}, I_{end})(IS) = \{int \in IS | start(int) < I_{end} \wedge end(int) \geq I_{start}\}$
- *StartingIn*: $SI[I_{start}, I_{end})(IS) = \{int \in IS | I_{start} \leq start(int) < I_{end}\}$
- *EndingIn*: $EI[I_{start}, I_{end})(IS) = \{int \in IS | I_{start} \leq end(int) < I_{end}\}$

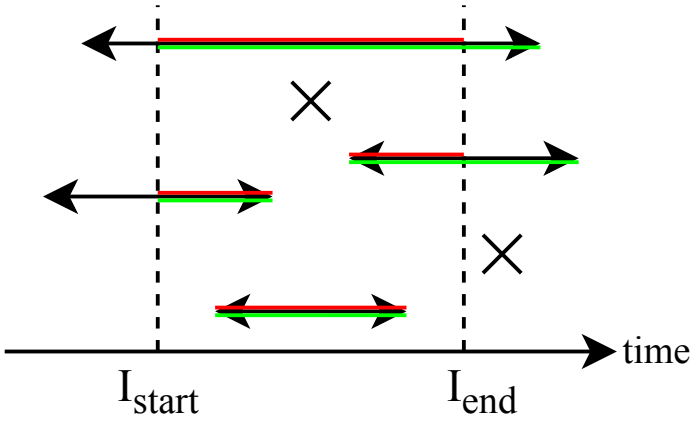


Figure 3.5: Different types of overlap of interactions with a time interval

- *ContainedIn*: $CI[I_{start}, I_{end})(IS) = \{int \in IS | I_{start} \leq start(int) \leq end(int) < I_{end}\}$

The actual overlap of a *Touching* interaction $int \in IS$ with the time interval is between $l\_limit = \max\{start(int), I_s\}$ to $r\_limit = \min\{end(int), I_e\}$, so

$$overlap[I_s, I_e)(int) = \begin{cases} r\_limit - l\_limit & \text{if } start(int) < I_e \wedge end(int) \geq I_s \\ 0 & \text{otherwise} \end{cases}$$

Figure 3.5 shows some example interactions. The complete ones have arrowheads giving the *start* and *end* times and the incomplete ones are represented by $\times$. All four complete ones are *Touching*. The second one is also *StartingIn*, the third one *EndingIn* and the last one is both of those which makes it *ContainedIn*. The top incomplete interaction is also everything while the bottom one is not touching at all. The overlap is colored red in the figure.

Finally, we define metrics for conformance, performance and process context. For conformance, we propose two similar local fitness measures. One is based on the ratio of complete interactions and all interactions, while the other counts events instead of whole interactions. Since complete interactions are a pair of two events, the latter will always be higher, but it considers the actual time of occurrence more exactly.

Definition 3.7 (Local fitness)**.** Let (CIS, IIS) be a pair of sets of complete and incomplete interactions and $[I_s, I_e)$ be a time interval. The interactions starting in the interval are $c\_start = SI[I_s, I_e)(CIS)$ and $i\_start = SI[I_s, I_e)(IIS)$ for complete and incomplete ones respectively.

$$lfitness_{int}[I_s, I_e)(CIS, IIS) = \begin{cases} \frac{|c\_start|}{|c\_start|+|i\_start|} & \text{if } |c\_start| + |i\_start| \neq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

The events from interactions inside the interval are $ce\_start = \{e | I_s \leq time(e) < I_e \wedge e \in \{e, e' | ((e,t),(e',t')) \in CIS\}\}$ and $ie\_start = \{e | I_s \leq time(e) < I_e \wedge ((e,t)) \in IIS\}$ for complete and incomplete interactions respectively.

$$lfitness_{event}[I_s, I_e)(CIS, IIS) = \begin{cases} \frac{|ce\_start|}{|ce\_start|+|ie\_start|} & \text{if } |ce\_start| + |ie\_start| \neq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

If no interactions/events occur in the chosen time interval, the fitness is undefined. For the interval in Figure 3.5, $lfitness_{int}[I_{start}, I_{end}) = 0.67$ and $lfitness_{event}[I_{start}, I_{end}) = 0.8$.

The current average sojourn time on a place is a measure for performance. The average is computed over complete interactions starting in the chosen interval. That means it is also undefined during intervals where no interactions occurred.

Definition 3.8 (Local performance)**.** Let CIS be a set of complete interactions and $[I_s, I_e)$ be a time interval. Again, let $c\_start = SI[I_s, I_e)(CIS)$.

$$lperf[I_s, I_e)(CIS) = \begin{cases} \sum_{ci \in c\_start} duration(ci)/|c\_start| & \text{if } |c\_start| \neq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

lperf describes how long (on average) cases that arrived at the place in the chosen interval eventually waited on there.

We propose three different metrics to try to measure the process context in terms of busyness at the current place during the chosen time interval. If many cases are currently waiting on this place, it will be busy.

Definition 3.9 (Local busyness)**.** Let CIS be a set of complete interactions and $[I_s, I_e)$ be a time interval.

- $lbusyness_{c\_int}[I_s, I_e)(CIS) = |SI[I_s, I_e)(CIS)|$
- $lbusyness_{activity}[I_s, I_e)(CIS) = \begin{cases} \sum_{ci \in CIS} overlap[I_s, I_e)(ci)/(I_e - I_s) & \text{if } I_s \neq I_e \\ \text{undefined} & \text{otherwise} \end{cases}$
- $lbusyness_{remsojourn}[I_s, I_e)(CIS) = \sum_{ci \in T[I_s, I_e)(CIS)} (end(ci) - \max(start(ci), I_s))$

$lbusyness_{c\_int}$ is the number of complete interactions starting in the interval. Incomplete interactions may also inhibit performance since they are also in the log and actually occurred, so a variant $lbusyness_{int}$ counting all interactions may also be useful. $lbusyness_{activity}$ is the sum of the ratios of overlap and the time interval size. The metric is relative to the interval size, so the interactions are weighed by their overlap which provides are more nuanced view on the busyness on a place during an interval. This is undefined for a time interval with the same start and end time. $lbusyness_{remsojourn}$ is the total remaining sojourn time from the start of the interval of all touching interactions, shown in green in Figure 3.5. In other words, if the place was a queue and could only handle one interaction after the other, it would take $lbusyness_{remsojourn}$ long to empty the queue. Instead of the total time, the average might also be interesting.

3.4 Tying it all together

The previous sections described the necessary process to arrive at metrics for conformance, performance and busyness. Given a system net $SN = (PN, M_i, M_f)$ and log $\mathcal{L}$, the set of complete CIS and incomplete IIS interactions can be extracted for each individual place $p \in P$. The metrics can then be calculated over time by dividing the whole process duration into a selected number of intervals or intervals of certain length like day, month or year. This yields a timeseries which can be used in statistical analysis to detect seasonality and trends. By computing the relative standard deviation (standard deviation

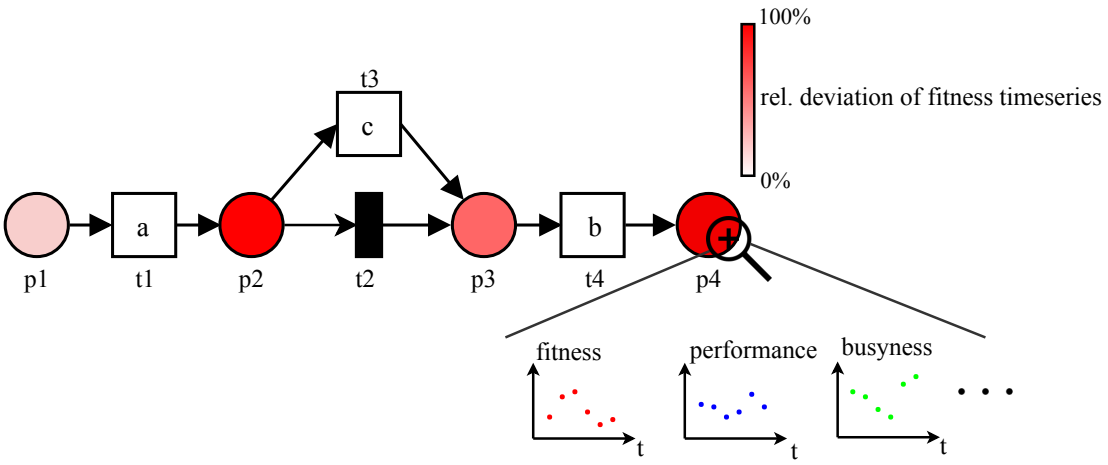


Figure 3.6: Schematic result of applying the approach

divided by mean) over the series or using other deviation metrics, one can also try to judge the stability of a certain metric over time. This way places can also be compared. Some might be in parts of the process that are quite streamlined and stable, while elsewhere the actual process execution may be more erratic in terms of conformance and performance. Figure 3.6 shows a schematic of this on the running example model. Another interesting application is finding correlations between these metrics. For example, does a high busyness value actually correlate with long sojourn times? In other words, how sensitive is this place to busyness? Or does a low conformance possibly lead to longer sojourn times? Since the *CIS* and *IIS* sets also contain the event and case data information, correlations can be extended to include those features. Which activity is often involved in incomplete interactions, and during which times? Do case attributes influence performance on this place? The localization of these questions to individual places and time frames makes it possible to find correlations which would be too weak to be detected over the whole log and process. For these purposes a dataset can be compiled containing all interactions with time and data information together with the metrics calculated for the duration of the interaction.

We also include another interesting aspect in our approach which is using the relative time since the case start. Every previously mentioned analysis can be performed for a relative time perspective by adjusting the intervals used for the metrics. It can be used to see after which time cases usually arrive at a place and if there are differences when they are later or earlier.

Chapter 4

Implementation

The entire approach has been implemented in the open source process mining framework ProM[1] as an interactive plugin in the package InteractiveLocalizedPlaceAnalysis. It takes as input an event log, which can be imported from many different data formats via the use of existing conversion plugins, and a Petri net model, with an initial and final marking connected to it. The interactive view is divided into two parts. On the left side, the process model is shown and on the right is a vertical tabbed pane. The first tab is the settings tab from which the algorithm can be started with different parameters. In addition to choosing if log moves should be mapped, is the option to map them but always treat them as incomplete interactions, to respect that they should not have been able to fire. The computation results are added as a new tab. Figure 4.1 shows one such tab with the different parts numbered (1)-(7). When selecting a place in the model, the result tab is updated asynchronously with statistics for that place. Switching to a different result tab at that point updates the stats again and makes it easy to compare the differences.

At the top of the result panel is a selection of perspectives (1) which determine the colorization of the model. The perspectives *conformance*, *performance*, *busyness* and *importance* color the places by ranking their respective metrics. The stability perspectives use the relative standard deviation of the metric for the currently selected discretization settings and color based on a capped scale from 0% to 100%. The discretization settings (2) have a slider to select the number of intervals and a toggle to choose if absolute time should be used or the relative time since case start.

Below that (3)-(6) is the place specific information. The first block of statistics (3) are the properties of the cases that have an interaction with the place. If the average of *Adjacent Case Events* is above 2, the place is involved in loops. The *Case Duration* is biased compared to the overall case duration average, so it can already be used to compare the performance of different paths in the model. *Sojourn Importance* is the ratio of the total sojourn duration of a case on this place to its entire case duration. The higher the value, the higher is the influence of this place's waiting time on the overall performance. (4) are the metrics defined in Section 3.3 calculated for each case over its whole duration and averaged. (5) are stats of the timeseries of these metrics for the entire place for the currently selected discretization settings of the process duration. The values here are those used in the colorization.

[1] http://promtools.org/

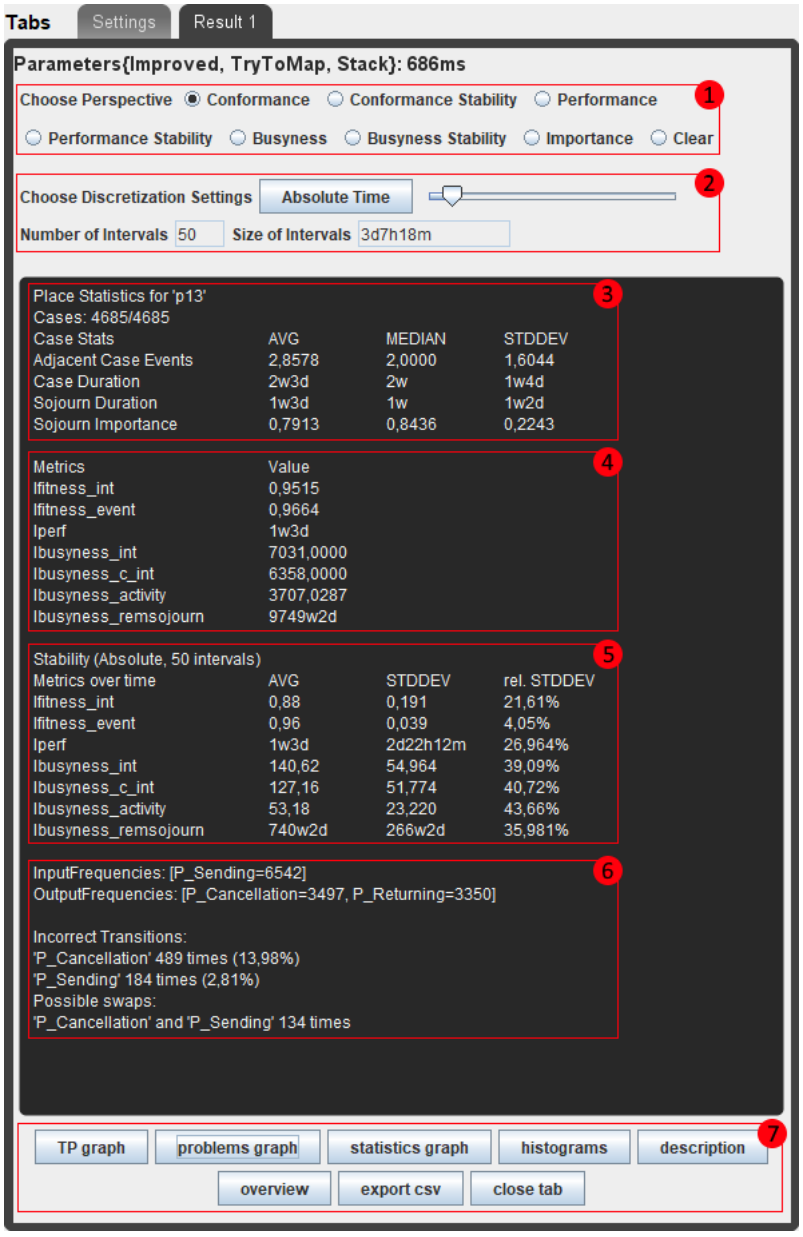


Figure 4.1: The main information view

The other info in this text panel (6) is about the mapped transition executions and how often they are in incomplete interactions. There is also a simple heuristic which tries to detect swapped events by checking if there are two incomplete interactions directly following one another where the first one is just an output transition and the second one an input transition. This obviously only works when log moves are mapped because these swaps cannot occur in sync moves.

Below the text panel are a number of action buttons (7). The *problems graph* shows four selected graphs concerning the conformance at the current place and is intended for visual correlation of the graphs. An example is shown in Figure 4.2. The top left graph displays the number of incomplete interactions per time interval. The top right shows the timeseries of $lfitness_{event}$ together with its standard deviation over all cases.

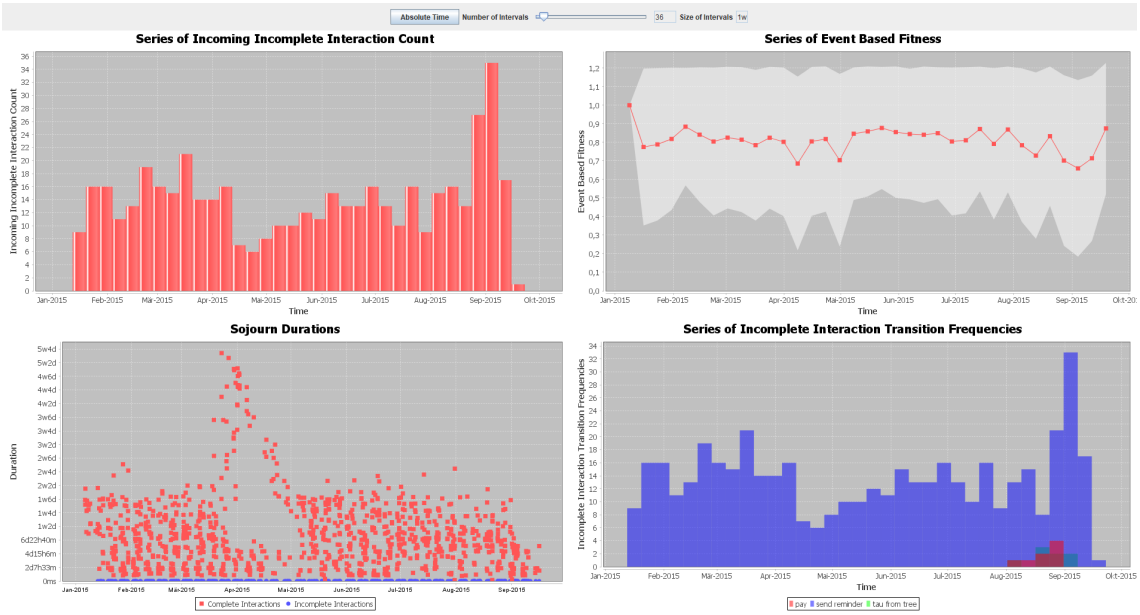


Figure 4.2: The problem view

In the bottom left is a scatterplot of interactions and their durations. The x-axis is the start time of the interaction and the y-axis the duration. Incomplete interactions are colored blue. The bottom right is a timeseries of the frequency of transition executions from incomplete interactions. It can be used to quickly diagnose which activity might be missing or occurring too often when the fitness is low.

The *statistic graph* is a generic view for displaying a graph of any of the presented metrics over time. There are even some more options to be able to experiment with the metrics and variants of them. It also has a button for quickly exporting such a timeseries for external statistical analysis. An example of it is Figure 4.3.

Additionally there is also an export button directly on the place information panel in (7). From there it is possible to export a compiled dataset of all interactions with the selected place enhanced with the localized metrics calculated for the duration of the interaction as discussed in Section 3.4. Table 4.1 shows an example for an extracted dataset. The exporter also offers an approximation option for big logs because of the quadratic time complexity of the exact calculations. For the approximation, the stats are calculated once for the selected number of intervals and then via the overlap of the interaction duration and intervals. There are a few other views using histograms and bar charts to present certain simple stats of cases interacting with the place. The *TP graph* button for example shows the graph of token counts of all locally mapped cases with token based replay on the selected place.

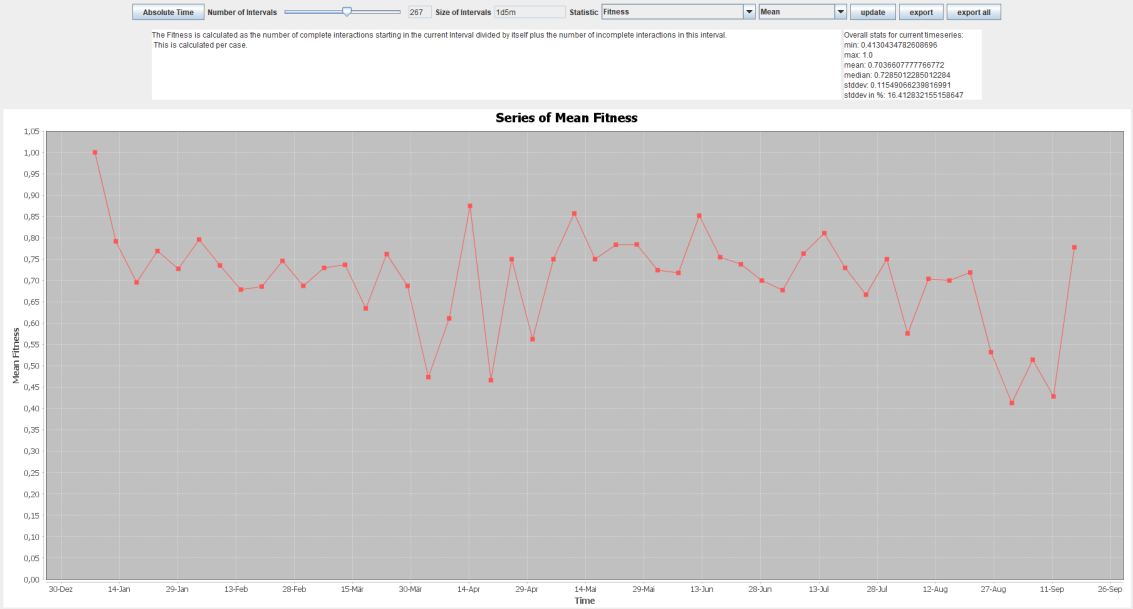


Figure 4.3: The generic statistic view

StartTime	CaseRelativeStartTime	SojournDuration	CaseDuration
2011-10-01 11:45	2s	1w1d	1w4d
2012-01-18 11:22	2s	4w2d	4w2d
2012-02-16 16:24	1s	0	1s
2012-02-14 20:38	9w6d	0	9w6d
lbusyness_int	lbusyness_activity	lbusyness_remsojourn	lperf
309.0	133.36	484w3d	1w4d
1752.0	541.87	2992w5d	1w4d
1.0	525.0	656w2d	
1.0	563.0	715w6d	
lfitness_int	lfitness_event	InputTransition	OutputTransition
0.99	0.99	O_SENT	O_SENT_BACK
0.90	0.94	O_SENT	O_CANCELLED
0.0	0.0	O_SENT	-
0.0	0.0	+	O_CANCELLED
IsComplete	Iteration	AMOUNT_REQ	concept:name
true	0	20000	173688
true	0	8000	201915
false	0	10000	205334
false	1	5000	191797

Table 4.1: Example for the exportable dataset

Chapter 5

Evaluation

To evaluate our approach, we first verify it on a synthetic log specifically generated to exhibit certain irregularities. Then we measure the performance with more synthetic logs and finally apply it to a real-life event log.

5.1 Synthetic log

The generated event log is made up of $10,000$ traces which are uniformly distributed over one year. The model used is shown in Figure 5.1. The baseline trace variant is perfectly fitting and has normally distributed sojourn times. One minute between a and b, one week between b and c and one day between c and d. Different types of *conformance* problems occur in three months of the year.

- *February* — b is skipped
- *April* — b is executed two times
- *June* — b and c are swapped

Furthermore, there are two *performance* problems concerning the sojourn duration between b and c.

- *August* — the sojourn duration is doubled
- *October* — the sojourn duration is halved

These deviations happen with a probability of 70% during the specified months.

Figure 5.4a on page 25 shows the result of *Replay a Log on Petri Net for Conformance Analysis*. The overall fitness is 0.97 and the problematic place is identified as the one before activity b. The detailed information is that b and c were involved in log moves about 600 times each while the token was on this place. With what we know about the log, this diagnosis is not sufficient.

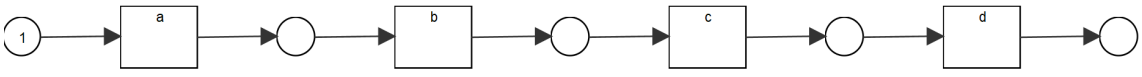


Figure 5.1: Simple sequential model

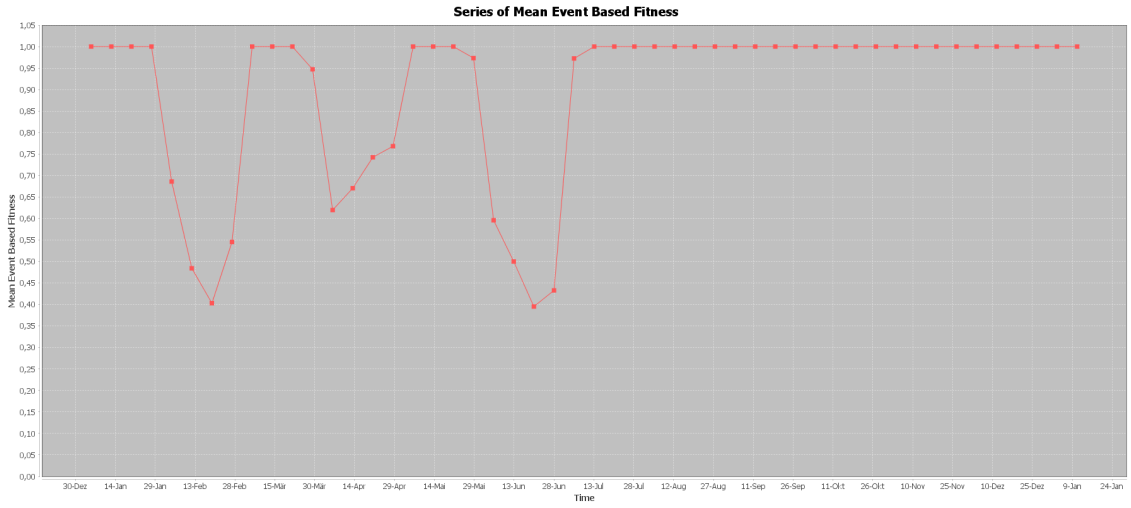


Figure 5.2: Fitness graph from our approach

The fitness graph over time from our approach for the place after b shown in Figure 5.2 is more insightful. There are three dips in fitness for the specified months. The dip for April is not as deep because the repetition of an activity still leaves one complete interaction and an incomplete one compared to skipping the activity entirely. Additionally, the swap heuristic on the result panel identifies the approximately 600 times b and c were swapped. The exportable dataset containing all interactions allows for a complete identification of the different problems but is out of the scope of the plugin.

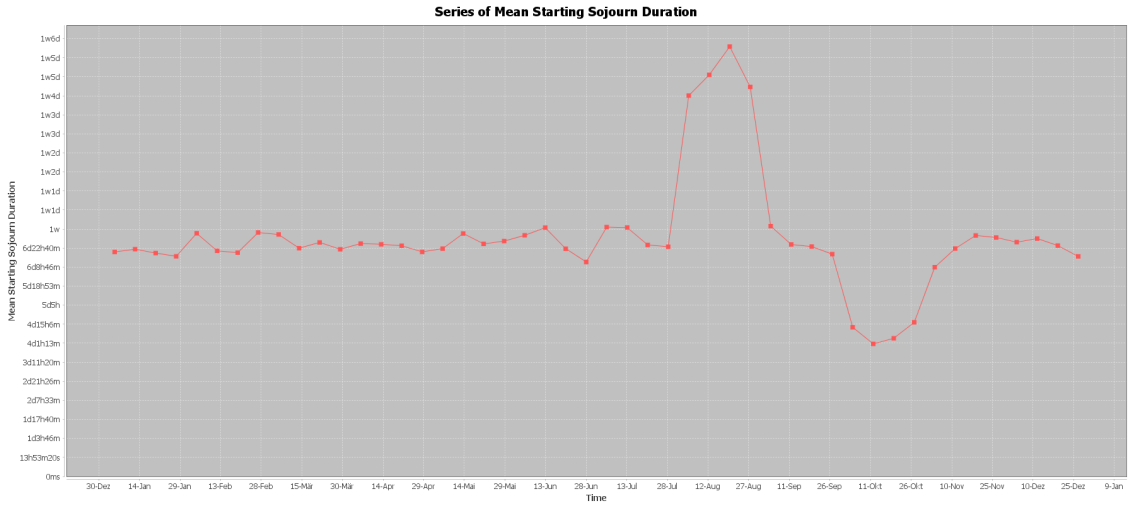
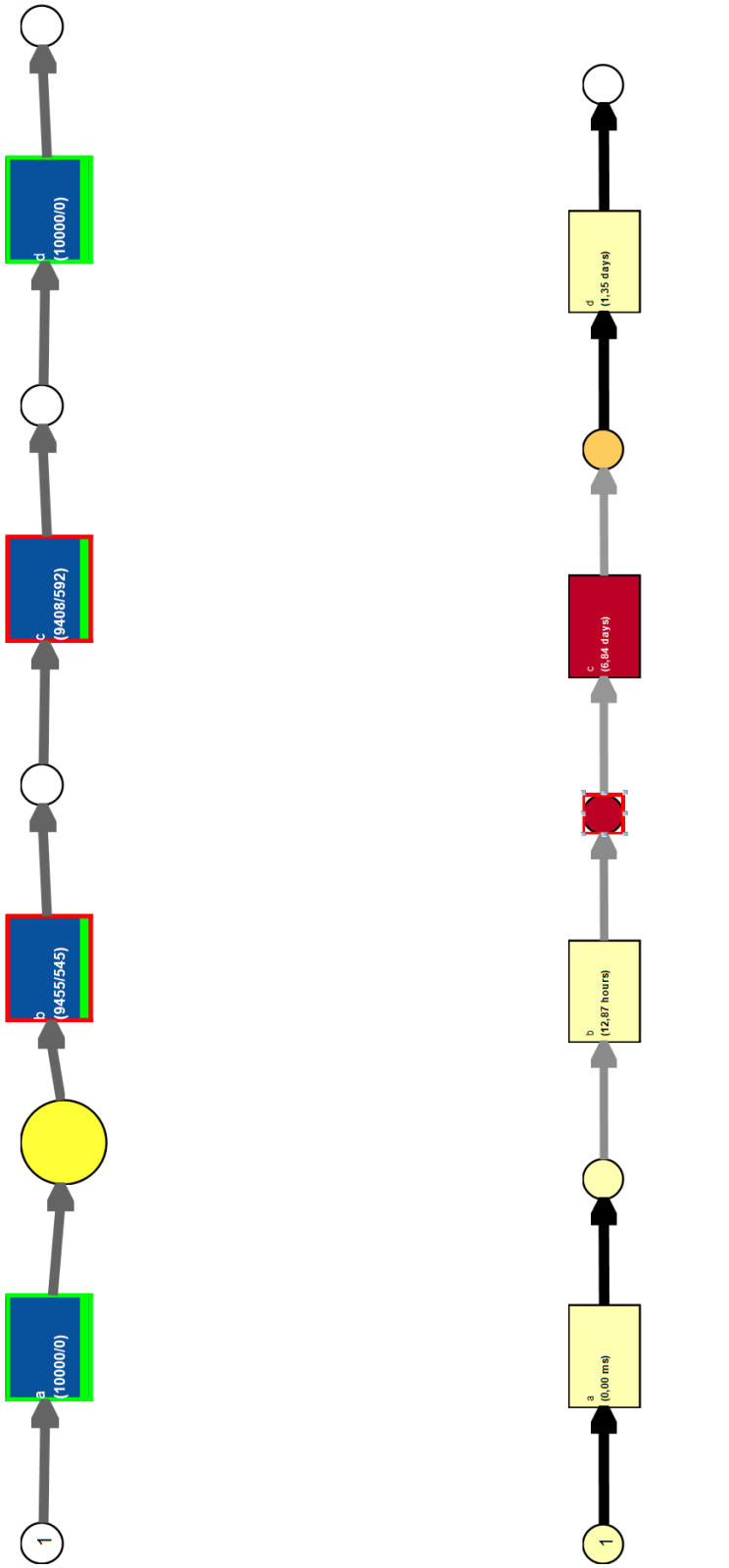


Figure 5.3: Performance graph from our approach

The performance view of plugin *Replay a Log on Petri Net for Performance/Conformance Analysis* shown in Figure 5.4b on page page 25 gives an average sojourn duration between b and c of 6.4 days with a standard deviation of 4.5 days. The standard deviation gives a hint towards the irregularities but nothing specific. Figure 5.3 on the other hand clearly shows the increase in August and the decrease in October based on *lperf*.



(a) Conformance view

(b) Performance view

Figure 5.4: State of the art results

5.2 Performance

We evaluate the performance of the plugin experimentally with synthetic event logs from [8]. The logs are simulated from randomized models which exhibit complex behavior with choices, loops and silent transitions. The execution times of the three different tasks: computing an alignment, mapping local logs and extracting interactions for all places are measured. The resulting averages with standard plugin settings on a machine with an *i5-3570* CPU are given below in Table 5.1.

log	cases	activities	events	alignment [ms]	local logs [ms]	interactions [ms]
1	256	16	985	32	3	1.33
2	1024	32	3951	281	5	5
3	4096	64	39306	3531	57	45
4	16384	128	107151	32141	647	89

Table 5.1: Performance on different log sizes

Alignment computation is very expensive but only has to be done once. Mapping the local logs is performed with Algorithm 1, so it is linear in the number of events but also dependent on the complexity of the model due to looping through the pre and postset. That makes it quadratic in the number of transitions in the worst case. This step is also parallelized over the number of available threads. Extracting the interactions for each place uses Algorithm 2, so it is linear in the output of the first step.

The relative durations of these tasks are shown in Figure 5.5. It becomes clear that computing the alignment is the dominant factor here. The bigger the log, the more negligible is the execution time of the approach itself. An improved *map_strat* not based on alignments could make this approach very scalable.

In general, the plugin computes most things asynchronously on-demand and caches many results for responsive interactive use.

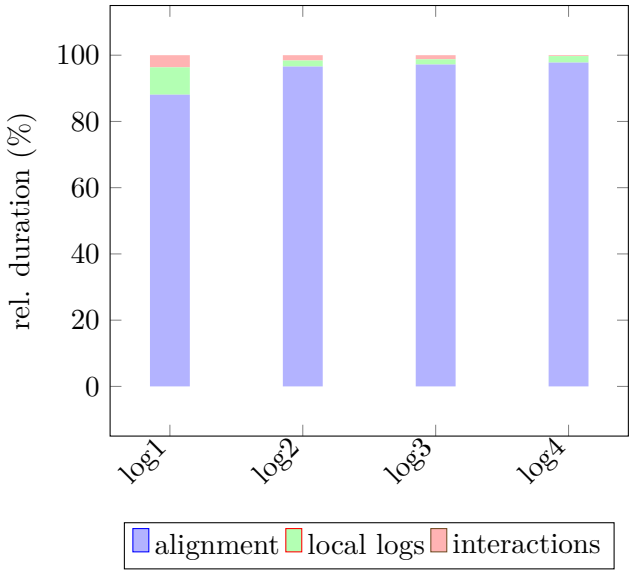


Figure 5.5: Relative execution times

5.3 Real-life log

Lastly, we apply the approach comparatively to a real-life event log about a loan application process from a financial institute. We show the advantages of our approach over a commercial tool and state-of-the-art ProM plugins in reasoning about conformance and performance. The log is from the BPI Challenge 2012 [19] and contains 262,200 events in 13,087 cases. It is actually composed of three intertwining subprocesses. Overall, cases start with an application, then multiple offers may be sent back and forth until the application is cancelled, declined or accepted. Work events pertaining to communication between employees and applicants are interspersed in between. Since this makes the process model discovery unfeasible, we focus just on the offer subprocess by projecting the entire log onto the offer activities. Since not every case includes an offer, this leaves 5015 cases which usually happen as follows: Offers are first selected, then created and sent. At this point, they are either sent back, and end up declined or accepted, or are cancelled and either end immediately or loop to the beginning. It is important to note here that the log was extracted from a running system and therefore contains incomplete cases.

Conformance

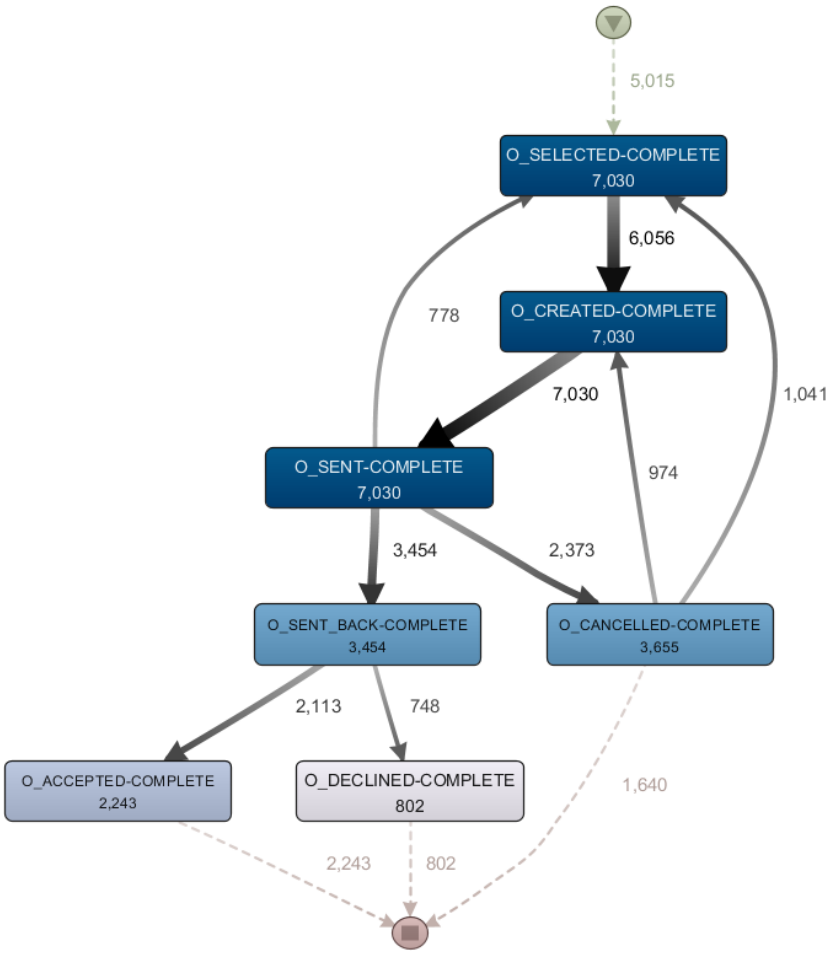


Figure 5.6: Adjusted process map of the commercial tool Disco

First, we apply the commercial tool *Disco*[1]. It uses an directly follows activity graph as a process model and cannot give direct insights into deviations because it is not executable. Due to that there is no notion of state and concurrency is not really supported. This already limits the ability to answer conformance questions. Disco includes a slider for filtering the directly follows relation by frequency which can be used to include rare deviations. Figure 5.6 shows this. This model incorporates the standard offer procedure but also includes new arcs which are hard to interpret.

Applying the state-of-the-art conformance checking plugin *Replay a Log on Petri Net for Conformance Analysis* is shown in Figure 5.10a on page 31. It yields a fitness value of 0.957 and places are sized relatively based on the number of times they are involved in non-sync moves. This means the per-place conformance has no real metric and is just comparative. The most deviations occur on the place before the split between O_CANCELLED and O_SENT_BACK. The plugin tells us that there are 767 log moves of O_SELECTED and 406 of O_SENT_BACK on that place. This alone does not allow diagnosing what exactly happened there and especially when.

We now apply our plugin to obtain more fine grained results.

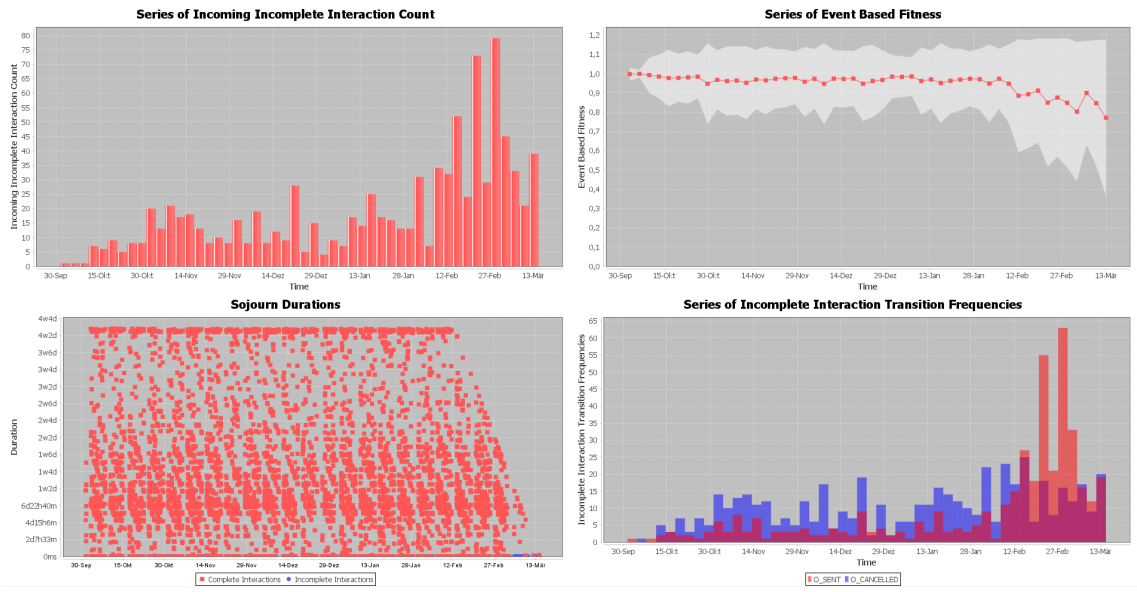


Figure 5.7: Problem view of the sink place

The problems view in Figure 5.7 of the place at the first split gives more insight into the deviations. The number of incomplete interactions increases towards the end, making the fitness drop. The regular sojourn durations also start to fade out after the 12th of February. We see that this is mainly caused by incorrectly executed event O_SENT, meaning cases simply ended afterwards. This is exactly one of the variants of incomplete cases. For the place before the O_SELECTED where the loop joins in again, the swap heuristic detects that an offer is sometimes selected before the loop, specifically the silent transition after the cancellation, is executed. This happens 578 times and is also exactly one of the common deviations from the model.

[1]https://fluxicon.com/disco/

Performance

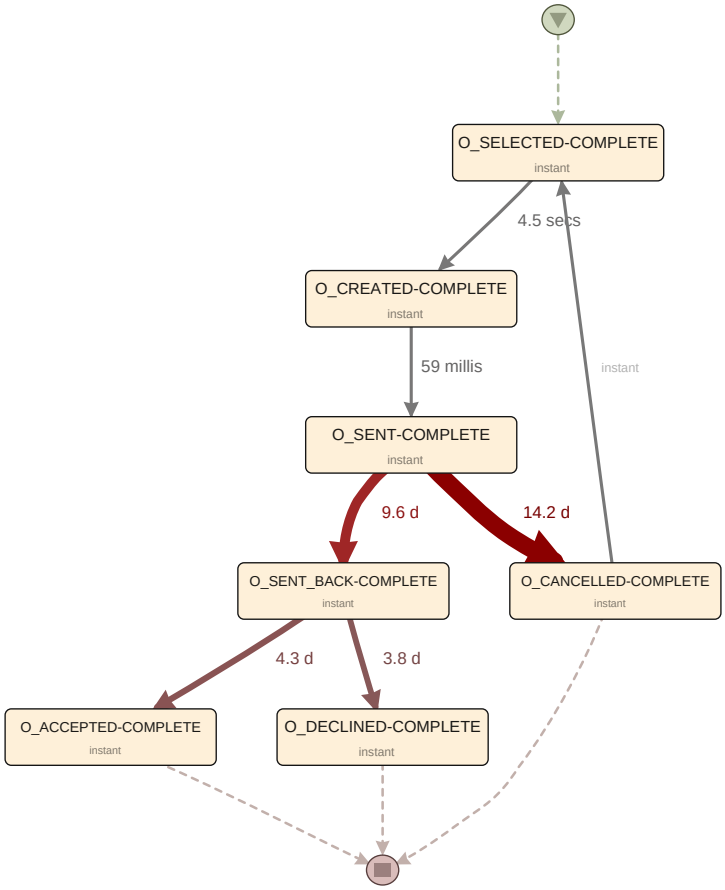


Figure 5.8: Performance view of the commercial tool Disco

Figure 5.8 shows the performance view of Disco on the standard model with average waiting times between activities. Due to not supporting concurrency, the numbers can be very misleading. There are a few different options apart from the average but a graph over time is not available. The closest one can get is filtering the cases with the extensive filtering options to compare more specific process maps and the new averages. This is never as fine as filtering the interactions over time.

The performance view of *Replay a Log on Petri Net for Performance/Conformance Analysis* in Figure 5.10b on page 31 is somewhat equally limited. The average sojourn time of the most problematic place which is the aforementioned first split is 10.62 days with a standard deviation of 9.61 days. Again, we can determine that the durations must be somewhat unstable, but we have no further information.

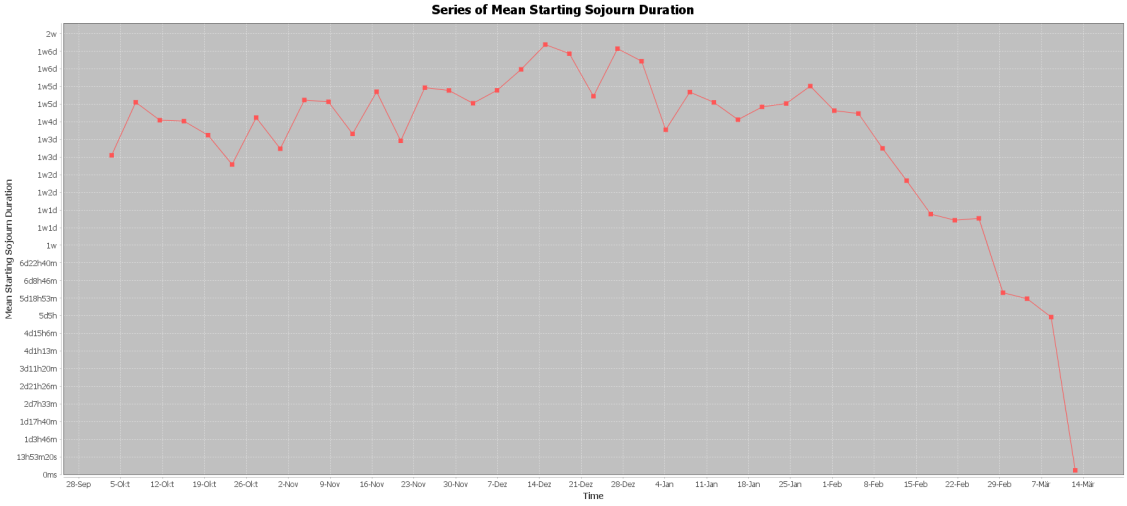
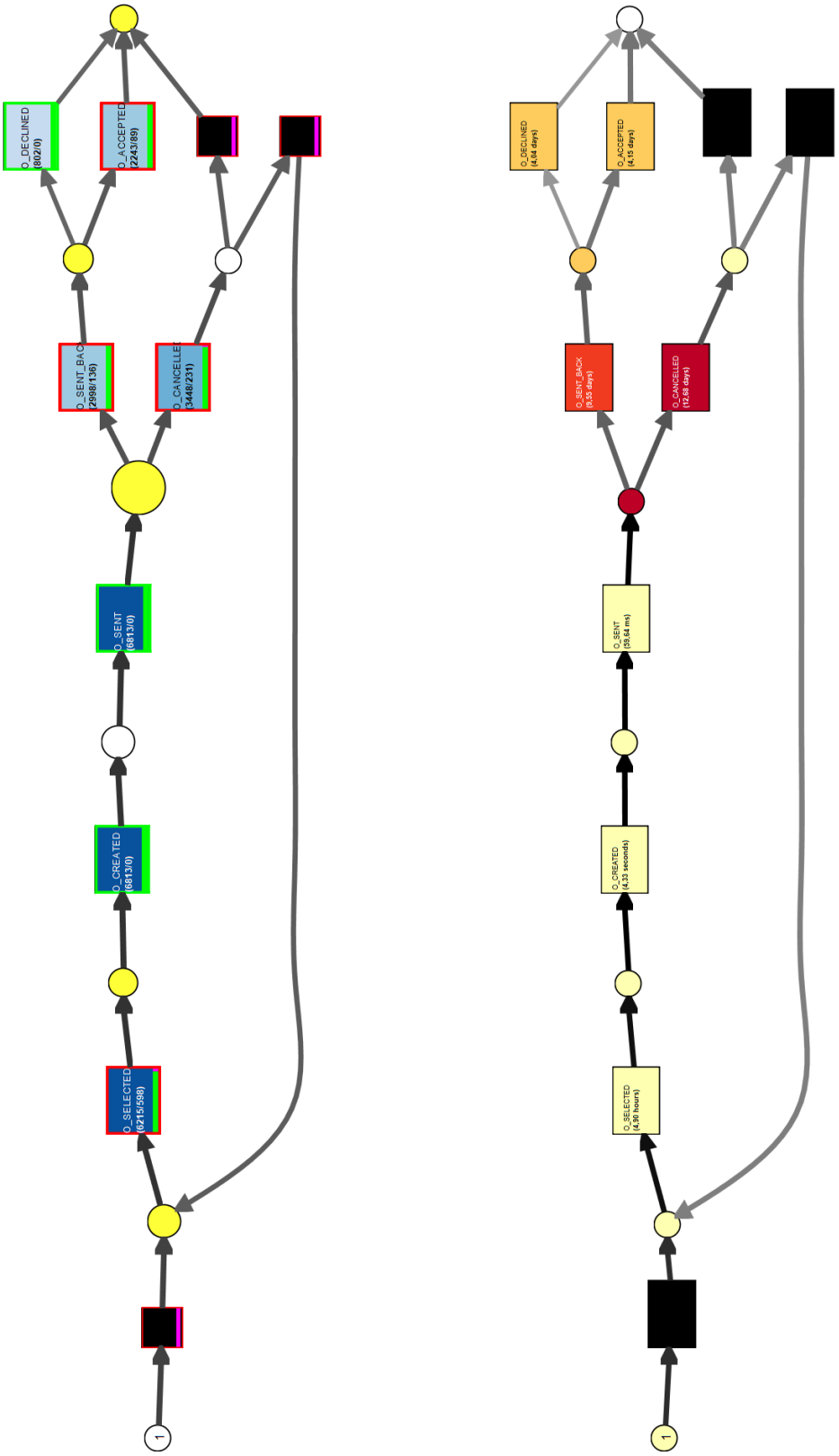


Figure 5.9: Sojourn duration graph over time

The performance graph for the interesting split place shown in Figure 5.9 clearly indicates that sojourn durations become much shorter towards the end of the observed timeframe. Together with the scatter plot in Figure 5.7, we can diagnose that there is an unnatural improvement in performance more likely caused by the falling fitness rather than actual improvements.

This evaluation showed the advantages of the most prominent features which are the timeseries of metrics. Especially the importance of localizing over time is apparent. An analysis of the exportable dataset is out of the scope of this evaluation but may lead to even more promising results.



(a) Standard conformance analysis (b) Standard performance analysis

Figure 5.10: Standard performance analysis

Chapter 6

Related Work

There are other notions of conformance other than fitness. [4] considers precision, generalization and simplicity in addition to fitness. Precision measures how well the model is restricted to just the actual observed behavior, i.e. testing if the model allows *more* than what happened in reality. This is explored extensively in [12]. Generalization is characterized by the chance that a the next incoming trace outside the training set also fits on the model. Simplicity measures the complexity and redundancy of the model which is important for human readability.

For fitness specifically, several different approaches are presented in [16]. Footprint based conformance compares the possible behavior of log and model through a directly follows abstraction. This measure takes into account fitness as well as precision, so it is not very specific. With token-based replay conformance checking, the number of produced, consumed, missing and remaining tokens are counted. This measures fitness but is influenced by the simplicity of the model because models with redundant places and the same behavior achieve a higher value. Lastly, alignment based conformance [1] is the state of the art approach to obtain stable fitness values. This is also the base for performance analysis. The events in the log are projected onto the synchronous moves of the alignment and used for calculating sojourn times and activity execution times. Our approach combines alignments with place-local token-based replay.

There are also various approaches dealing with a localization over control-flow. One application is decomposition of the process model. In [13] a generic concept to decompose discovery and conformance checking into smaller problems to increase efficiency is introduced. The log is projected onto overlapping sets of activities. This way, the results can be recombined afterwards. Since most algorithms for discovery and conformance checking are linear in the number of traces and exponential in the number of activities, an exponential speedup can be achieved even on a single core machine. A framework supporting multiple discovery algorithms has been implemented in [20]. For decomposed conformance checking, the implementation in [7] provides exact alignment based fitness values and strong guarantees for time-bound approximations by using re-composition to merge partial results. A novel approach also using a localization to places in discovery is [17]. Here, the model discovery is entirely decomposed to individual places. Places are viewed as a pair of ingoing arcs and outgoing arcs to transitions. Based on these sets, a partial order over possible places is defined. It ranks the ability of a place to fill with tokens. A light place has few input arcs and many output arcs. A heavy place has many input arcs and fewer output arcs. A variety of quality measures for candidate places is

defined and monotonicity over this partial order is proven. Based on this, many ways of pruning the search space are demonstrated in a generic way. Lastly, an application to efficient conformance checking is proposed.

Localization over the time dimension is used in streaming model discovery approaches which try to deal with concept drift. In [3], different features like the causal footprint of the log, which is derived from the directly-follows relation of activities, are measured over time and used to detect change points with statistical hypothesis tests. Then, a model is discovered for each section between change points. [11] also uses both localization over control-flow and time in the log to detect concept drift via statistical testing.

Considering multiple dimensions is also possible by clustering the log along chosen dimensions at first and then using conventional tools on the sublogs to obtain fine grained results. This is done in [6] with comparative trace clustering. After clustering, the models for the sublogs can provide more insight into seemingly unstructured processes. [5] proposes a general framework for case clustering while allowing for additional, user-defined, features to be added to events. This way, even more perspectives on the log may be considered.

Approaches combining several perspectives for diagnosing log and model have also been proposed. The multi-perspective process explorer [9] building onto [10], provides views for conformance, performance and data-flow. The alignments in [10] also take into account multiple dimensions like data, making each dimension a first-class citizen of the alignment, to allow more balanced conformance checking that is not dominated by control-flow.

In contrast to existing conformance and performance related approaches, we use localization over time in addition to control-flow to allow users to quickly compare and explore detailed metrics. Especially the timeseries functionality is an improvement over the state-of-the-art. Additionally, we provide a process context perspective and the opportunity for clustering and correlation with the exportable dataset.

Chapter 7

Conclusion

Real-life processes are rarely static over time. Deviations and performance problems may follow certain temporal patterns. Also, the places in a Petri net can behave very differently. Some places might always be stable, while others may exhibit more fluctuating behavior. To allow perspectives like conformance and performance to be fine grained enough, the analysis needs to be localized over multiple dimensions. In this approach, we first project the log on individual places, i.e. localizing over control flow. Then, the transition executions on a place are paired up with a heuristic, to extract complete and incomplete interactions. Since they have defined start and end times, they are further localized over time and associated to time intervals. For these intervals, we define metrics for conformance, performance and busyness, as a form of local process context. This allows a much finer analysis because averages tend to obscure lots of valuable information. Additionally, we provide a dataset made up of all interactions together with the localized metrics for their duration.

To evaluate our approach, we applied it comparatively to a purpose-generated synthetic log and a real-life log. We showed that timeseries of metrics can be of great help when diagnosing conformance and performance. Especially compared to the standard alignment-based plugins, the negligible additional computing time makes using our plugin worthwhile. A drawback however, is that we do not support activity execution times based on lifecycle information of the events. This perspective is still invaluable for a thorough performance analysis.

As future work, the approach could be parallelized further to reduce computation time. It could even be implemented in a map-reduce fashion for distributed computing. As it is, scalability is limited by the usage of alignments which we noted in Section 5.2. To remedy this, a different way of handling silent and duplicate transitions could be explored or results of decomposition could be applied, since the approach is localized to individual places anyway. Another drawback of our implementation is that it only supports case attributes and not event attributes which are even more appropriate for such a localized concept. And lastly, an encompassing analysis of the exportable dataset could be performed. Especially concept drift detection and clustering may be applied here.

Bibliography

[1] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F van Dongen, and Wil MP van der Aalst. Measuring precision of modeled behavior. *Information systems and e-Business Management*, 13(1):37–67, 2015.

[2] Thomas Baier, Claudio Di Ciccio, Jan Mendling, and Mathias Weske. Matching of events and activities-an approach using declarative modeling constraints. In *International Conference on Enterprise, Business-Process and Information Systems Modeling*, pages 119–134. Springer, 2015.

[3] RP Jagadeesh Chandra Bose, W.M.P. van der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. Handling concept drift in process mining. In *International Conference on Advanced Information Systems Engineering*, pages 391–405. Springer, 2011.

[4] Joos C.A.M. Buijs, Boudewijn F Van Dongen, and W.M.P. van Der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 305–322. Springer, 2012.

[5] Massimiliano De Leoni, W.M.P. van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235–257, 2016.

[6] B.F.A. Hompes, Joos C.A.M. Buijs, W.M.P. van der Aalst, P.M. Dixit, and Johannes Buurman. Detecting changes in process behavior using comparative case clustering. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 54–75. Springer, 2015.

[7] Wai Lam Jonathan Lee, H.M.W. Verbeek, Jorge Munoz-Gama, W.M.P. van der Aalst, and Marcos Sepúlveda. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Information Sciences*, 466:55–91, 2018.

[8] S.J.J. (Sander) Leemans. A collection of artificial event logs to test process discovery and conformance checking techniques, 2017.

[9] Felix Mannhardt, Massimiliano De Leoni, and Hajo A Reijers. The multi-perspective process explorer. *BPM (Demos)*, 1418:130–134, 2015.

[10] Felix Mannhardt, Massimiliano De Leoni, Hajo A Reijers, and W.M.P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016.

[11] M.V. Manoj Kumar, Likewin Thomas, and B Annappa. Capturing the sudden concept drift in process mining. *Algorithms & Theories for the Analysis of Event Data (ATAED'15, Brussels, Belgium, June 22-23, 2015)*, page 132, 2015.

[12] Jorge Munoz-Gama et al. *Conformance checking and diagnosis in process mining.* Springer, 2016.

[13] W.M.P. van der Aalst. Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[14] W.M.P. van der Aalst. Process mining in the large: a tutorial. In *European Business Intelligence Summer School*, pages 33–76. Springer, 2013.

[15] W.M.P. van der Aalst. Extracting event data from databases to unleash process mining. In *BPM-Driving innovation in a digital world*, pages 105–128. Springer, 2015.

[16] W.M.P. van der Aalst. *Process mining: data science in action.* Springer, 2016.

[17] W.M.P. van der Aalst. Discovering the "glue" connecting activities. In *It's All About Coordination*, pages 1–20. Springer, 2018.

[18] W.M.P. van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[19] B.F. Van Dongen. Bpi challenge 2012, 2012.

[20] H.M.W. Verbeek, W.M.P. van der Aalst, and J Munoz-Gama. Divide and conquer: A tool framework for supporting decomposed discovery in process mining. *The Computer Journal*, 60(11):1649–1674, 2017.