PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

ADVANCING DECOMPOSED CONFORMANCE CHECKING IN PROCESS MINING

WAI LAM JONATHAN LEE

Thesis submitted to the Office of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor in Engineering Sciences

Advisor:
JORGE MUÑOZ GAMA

Santiago de Chile, July 2020

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

ADVANCING DECOMPOSED CONFORMANCE CHECKING IN PROCESS MINING

WAI LAM JONATHAN LEE

Members of the Committee:
JORGE MUÑOZ GAMA
DENIS PARRA
MARCOS SEPÚVEDA
MARÍA CECILIA BASTARRICA
WIL VAN DER AALST
GUSTAVO LAGOS

Thesis submitted to the Office of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor in Engineering Sciences

Santiago de Chile, July 2020

Gratefully to my parents and brother

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the recent years, process mining has been gaining traction as a tool for analyzing and improving processes in the industry as exemplified by companies such as Disco, Celonis, and Minit. Furthermore, many commercial process mining tools are extending beyond process discovery to conformance checking that allows stakeholders to compare observed and modeled behavior to find discrepancies between how they expect their processes to be executed and how their processes have actually been executing. Performing conformance checking in industrial settings means that techniques have to be able to address the different data dimensions. For example, conformance checking techniques have to scale from processes of small companies to multinational organizations that may be handling many cases per hour. This thesis focuses on conformance checking and specifically addresses the challenges arising from the application of conformance checking in different scenarios.

Alignment-based techniques are the state of the art for identifying and explaining discrepancies between observed and modeled behavior. However, due to the explosion of state-space with processes with parallel constructs, alignment can be computationally expensive. The first part of the thesis focuses on extending decomposition techniques to alignment computation. The thesis shows that alignment can be computed in a decomposed manner and presents a novel conformance checking framework that computes alignment using the divide and conquer paradigm. There are now many conformance checking techniques available for end users. However, it can be difficult to select the best algorithm for the job since this depends on the input data and the user's objective. The second part of the thesis investigates machine learning techniques to help users select the best algorithm depending on their input data. Specifically, it applies machine learning to the classification problem of whether if decomposition techniques can improve computation time given the input model and log. The third part of the thesis turns to online conformance checking.

Given the volume and velocity at which event data comes, organizations may not store all the generated data for offline analysis and instead have to resort to online techniques. Moreover, performing analysis in real time allows process stakeholders to react to conformance issues before it is too late. Performing conformance checking in an online settings has its own unique challenges. For example, the conformance checking technique has to balance between putting emphasis on the current information and ensuring that the conformance result is somewhat stable as the running case unfolds. The thesis presents a novel online conformance checking technique based on Hidden Markov Models that focuses on this challenge.

Keywords: process mining, BPM, conformance checking, online processing, decomposition

RESUMEN

En los últimos años, la “process mining” ha ido ganando terreno como herramienta para analizar y mejorar los procesos en la industria, como lo ejemplifican empresas como Disco, Celonis y Minit. Además, muchas herramientas comerciales de process mining se están extendiendo más allá del descubrimiento de procesos a la “conformance checking” que permite comparar el comportamiento observado y modelado para encontrar diferencias entre los dos. Realizar la conformance checking en entornos industriales significa que las técnicas deben poder abordar las diferentes dimensiones de los datos. Por ejemplo, las técnicas de conformance checking tienen que escalar desde procesos de pequeñas empresas hasta organizaciones multinacionales que pueden estar manejando muchos casos por hora. Esta tesis se centra en la conformance checking y aborda especficamente los desafóos que surgen de la aplicación de la conformance checking en diferentes escenarios.

Las técnicas basadas en alineación (alignment) son el estado del arte para identificar y explicar las discrepancias entre el comportamiento observado y modelado. Sin embargo, debido a la explosión del espacio de estados con procesos con construcciones paralelas, la alignment puede ser computacionalmente costosa. La primera parte de la tesis se centra en extender las técnicas de descomposición al cálculo de alignment. La tesis muestra que la alignment se puede calcular de forma descompuesta y presenta un algoritmo de conformance checking novedoso que calcula la alignment utilizando el paradigma de dividir y conquistar. En la actualidad, existen muchas técnicas de conformance checking disponibles para los usuarios. Sin embargo, puede resultar difcil seleccionar el mejor algoritmo para el trabajo, ya que depende de los datos de entrada y del objetivo del usuario. La segunda parte de la tesis investiga técnicas de machine learning para ayudar a los usuarios a seleccionar el mejor algoritmo en función de sus datos de entrada. Específicamente, aplica el machine learning al problema de clasificación de si las técnicas de descomposición pueden mejorar el tiempo de cálculo dado el modelo y el event log. La tercera parte de la

tesis se centra en la conformance checking en tiempo real. Dado el volumen y la velocidad a la que llegan los datos de eventos, es posible que las organizaciones no almacenen todos los datos generados para el análisis offline y, en su lugar, tengan que recurrir a técnicas en tiempo real. Además, realizar análisis en tiempo real permite a los dueños del proceso reaccionar y resolver los problemas en forma inmediato. Realizar la conformance checking en una configuración en tiempo real tiene sus propios desafíos únicos. Por ejemplo, la técnica de conformance checking tiene que equilibrar el énfasis en la información actual y garantizar que el resultado de conformance sea algo estable a medida que se desarrolla el caso en ejecución. La tesis presenta una novedosa técnica de conformance checking en tiempo real basada en Hidden Markov Model.

Part I

Introduction

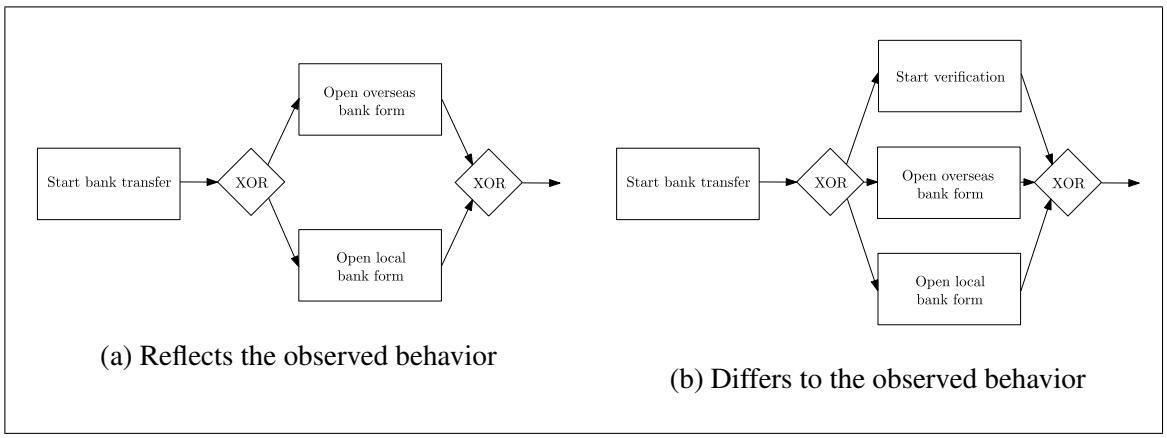
1. Introduction

Nowadays, large amount of information is being recorded by organizations during their daily operations. For example, Wal-Mart is estimated to collect more than 2.5 petabytes of data every hour from its customer transactions [27]. Other than to support their operations, business stakeholders are actively using data to drive their decision making [66]. Process mining applies data-driven approaches to analyze, improve and manage processes. While it is not strictly limited to business processes, process mining is typically positioned within the scope of business process management (BPM) due to much of its existing work. By bringing a data perspective, process mining adds to the traditionally model-driven approaches in BPM.

More and more event data are made available with the increased adoption of information systems [2]. This means that process mining techniques have to be able to handle the analysis of large processes under various contexts. In this chapter, we present about the challenges that existing techniques are facing and give an overview of the contributions of this thesis towards the identified challenges. We first provide a brief introduction to process mining and the problem of conformance checking as the main area of process mining that this thesis focuses on.

1.1. Process mining

Information systems such as ERP (Enterprise Resource Planning) systems (SAP, Oracle, etc) and BPM (Business Process Management) systems (Pegasystems, Bizagi, Appian, IBM BPM, etc) support processes in different organizations. Event data from the event logs of information systems can be ordered to describe instances of the underlying process such that each event can be related to an activity and belongs to a particular case of the process. In essence, these event logs can be seen as “footprints” left behind by the execution of the process. Process mining uses these event logs as a starting point to “discover, monitor and improve real processes” [66].



(a) Reflects the observed behavior

(b) Differs to the observed behavior

Figure 1.1. Different XOR choice constructs of activities observed in event log at Table 1.1

Consider a bank transfer process in which a client makes a bank transfer from one account (sender account) to another account (receiver account) which can be either a local or an oversea bank. The process may begin by a bank operator initiating a new bank transfer in the system (Start bank transfer). Then the operator would fill in the corresponding forms and verify the client's account and the receiver's account before completing the bank transfer. Table 1.1 shows an example of an event log, organized after extracting the relevant information from the information system. Each row represents an event of the log, and events can be grouped by their case ID to form instances of the process.

There are three main areas in process mining [66]. *Process discovery* takes an event log as input and produces a model using different discovery algorithms. For example, considering the event log in Table 1.1, one can see that after starting the bank transfer, the operator can either open up an overseas bank form or open up a local bank form. This part of the process can be visualized as an XOR choice construct as shown in Figure 1.1a. In simple words, process discovery summarizes the relations between activities as seen in observed data and represent these relations as a graphical construct.

In process discovery, at times, one does not want to include all of the observed behavior into the discovered model since that might result into a spaghetti-like model that is impossible to understand. At other times, one might want their discovered model to

Case	Event	Timestamp	Activity	Employee	Client
1	1	2020-01-01 09:00:00	Start bank transfer	Jim	Anna
1	4	2020-01-01 09:10:00	Open overseas bank form	Jim	Anna
1	6	2020-01-01 09:15:00	Enter oversea bank code	Jim	Anna
1	7	2020-01-01 09:23:00	Foreign currency conversion	Jim	Anna
1	9	2020-01-01 09:25:30	Update oversea bank form	Jim	Anna
1	10	2020-01-01 09:30:00	Start verification	Tom	Anna
1	12	2020-01-01 09:31:10	Enter sender account	Tom	Anna
1	14	2020-01-01 09:35:00	Verify sender account	Tom	Anna
1	15	2020-01-01 09:40:00	Enter receiver account	Tom	Anna
1	18	2020-01-01 09:45:00	Verify receiver account	Tom	Anna
1	19	2020-01-01 09:48:00	Complete verification	Tom	Anna
1	21	2020-01-01 09:50:00	Finish bank transfer	Jim	Anna
1	23	2020-01-01 09:55:00	Send bank transfer	Jim	Anna
2	2	2020-01-01 09:02:00	Start bank transfer	Jerry	William
2	3	2020-01-01 09:08:00	Open local bank form	Jerry	William
2	5	2020-01-01 09:13:00	Enter local bank code	Jerry	William
2	6	2020-01-01 09:20:00	Update local bank form	Jerry	William
2	8	2020-01-01 09:25:00	Start verification	Jerry	William
2	11	2020-01-01 09:30:20	Enter receiver account	Jerry	William
2	13	2020-01-01 09:34:00	Verify receiver account	Jerry	William
2	16	2020-01-01 09:41:00	Enter sender account	Jerry	William
2	17	2020-01-01 09:43:00	Verify sender account	Jerry	William
2	20	2020-01-01 09:49:00	Complete verification	Jerry	William
2	22	2020-01-01 09:53:00	Finish bank transfer	Jerry	William
2	24	2020-01-01 09:58:00	Send bank transfer	Jerry	William
3	25	2020-01-03 10:00:00	Start bank transfer	Joanne	Bobby
3	26	2020-01-03 10:12:00	Open overseas bank form	Joanne	Bobby
3	27	2020-01-03 10:15:00	Open local bank form	Joanne	Bobby
...	...	...	...	...	...

Table 1.1. Example of event log for bank transfer process

include behavior that is unobserved but foreseeable to occur in the future. With different discovery algorithm being proposed, the need of being able to compare the observed behavior in the event log and the modeled behavior in the discovered model came about. Conformance checking compares an event log with an existing process model of the same process to identify commonalities and discrepancies. For example, it is clear that the XOR construct shown in Figure 1.1b have discrepancies with the behavior observed in the event

log at Table 1.1. The activity *Start verification* seems to be a necessary step rather than a choice amongst the options of opening overseas and local bank forms. A conformance checking algorithm would identify such a difference.

Lastly, information from recorded data can also be aggregated to a process model. *Enhancement* enriches an existing process model using information from recorded event logs. This thesis focuses on **conformance checking**.

1.2. Conformance checking

There are different reasons for performing conformance checking [17]. For example, it can be integrated into the audit of business processes [66, 48]. In businesses, there are often protocols set by different stakeholders, e.g., managers, government, and others. Audits are carried out to verify that these internal and external regulations are complied. By comparing the events recorded during process execution and models describing business protocols, it is clear that conformance checking can help and support the automation of audits [70, 69, 29]. Other than ascertaining business compliance, process stakeholders may also want to know whether if the current protocols are meeting the requirements of its execution in a real-life context. In this sense, conformance checking can help with process re-design and modification of the BPM lifecycle [24]. Finally, conformance checking is essential within process mining to compare the quality of discovery algorithms [22], to recommend discovery algorithms [55], and even to be integrated into discovery algorithms [21, 10].

1.2.1. Conformance quality dimensions

There are four quality dimensions for measuring conformance between an event log and a process model [66, 48].

Fitness measures the amount of observed behavior that is modeled by the process model. For example, the subtrace of case 3 in the partial event log at Table 1.1 ($\langle$Start bank transfer,

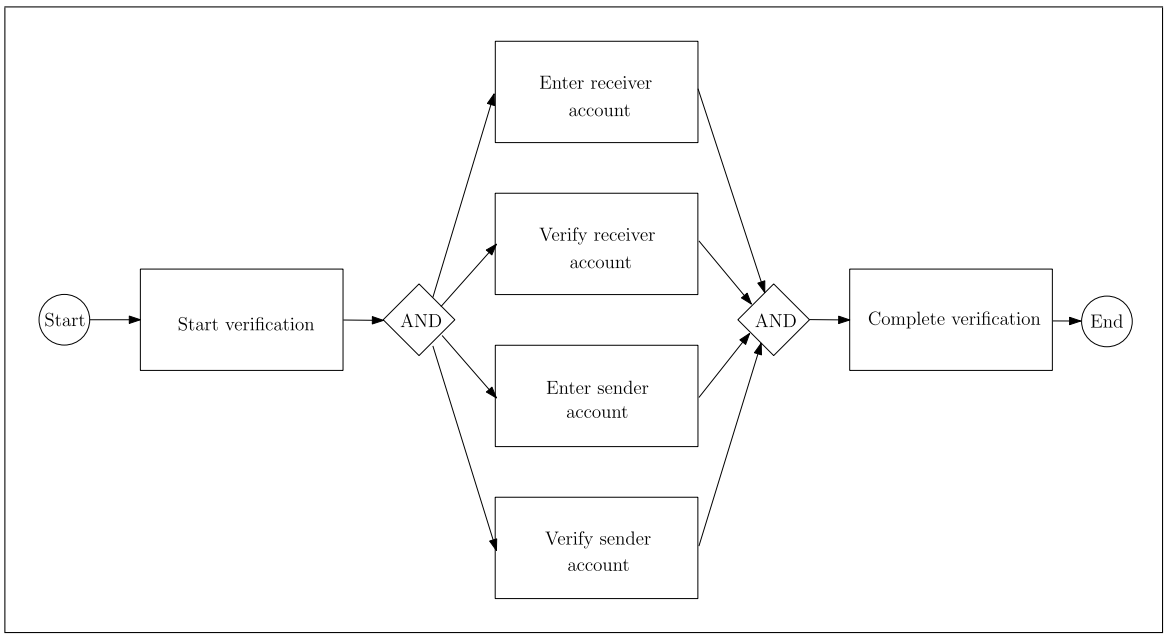


Figure 1.2. Possible model of verification subprocess

Open overseas bank form, Open local bank form⟩) is clearly not fitting with the XOR construct shown in Figure 1.1a. The operator is supposed to either open up an overseas bank form or open up a local bank form but not both.

Precision measures the amount of modeled behavior that is actually observed in the event data. For example, lets look at the verification process which includes the activities (Start verification, Enter receiver account, Verify receiver account, Enter sender account, Verify sender account, Complete verification). Figure 1.2 shows a possible model of the process. This model is clearly fitting with the event log in Table 1.1 since the sequences (⟨Start verification, Enter sender account, Verify sender account, Enter receiver account, Verify receiver account, Complete verification⟩, and ⟨Start verification, Enter receiver account, Verify receiver account, Enter sender account, Verify sender account, Complete verification⟩) are both possible in the model. However, the model is imprecise since it also allows for unseen behavior. For example, the model allows for the sequences (⟨Start verification, Enter receiver account, Enter sender account, Verify sender account, Verify receiver account, Complete verification⟩, and ⟨Start verification, Enter sender account, Enter receiver account,

Verify sender account, Verify receiver account, Complete verification$\rangle$) which are not observed in the event log.

Generalization measures whether if the model is too strict so that only observed behavior is permitted in the model. In general, only a small percentage of possible behavior might be observed in reality, especially with parallel constructs. This means that it is desirable for the model to generalized beyond the observed behavior to include behavior that is not observed but foreseeable to occur in the future. In this case, the model in Figure 1.2 would score better in terms of generalization than precision.

Simplicity measures whether if the model is not overly complicated. Typically this quality dimension is in isolation of the event log.

1.3. Challenges

Conformance checking has become a topic of interest, not only academic but also commercially, where several process mining commercial tools such as Celonis and Lana Labs have recently incorporated preliminary conformance analysis.

Moreover, as process mining tools in the industry mature, the characteristics of incoming event data used for process mining can vary a lot. For example, in the past, the workflow of a process miner might consist of writing a custom script that pulls in data of a time period and creates a case notion so that it can be saved as in the XES format [3]. This event log file would then be used for various process mining analysis. Nowadays, commercial process mining tools, e.g. Celonis, can perform ETL (Extract, Transform, Load) operations directly on existing data sources, e.g., Salesforce and SAP, so that process mining results are immediately available.

This means that there is a need to address the challenges arising from the possible varying characteristics of input data to process mining algorithms. One popular way of

categorizing these challenges is by the different dimensions of data - the "four V's of data": *Volume*, *Velocity*, *Variety*, *Veracity* [33].

The first "V" (Volume) refers to the incredible amount of available data. The second "V" (Velocity) refers to the rate at which data is coming in. The third "V" (Variety) refers to the different forms in which data is being made available. The fourth "V" (Veracity) refers to the different degrees of trustworthiness of the data.

Examining these different data dimensions and conformance checking, we identify four challenges: *computationally expensive techniques*, *decomposition techniques*, *multitudes of algorithms for the same task*, and *online conformance checking*. This thesis addresses all of these challenges.

Challenge 1: *Computationally expensive techniques.* Good conformance checking algorithms goes beyond replaying a given observed log trace on the process model to verify if it is conforming. They may focus on specific conformance quality dimensions, as exemplified by techniques that focus on fitness [5, 6], precision [42, 47], generalization [76]. They may also try to yield robust explanation and diagnosis on the conformance issues [49, 4, 6] or focus on tackling specific data type and conformance issues [44, 9]. However, a fundamental problem in conformance checking is that the number of possible process instances as described by a process model often increases exponentially with respect to the number of activities. This means that sophisticated techniques that look at a large amount of modeled behavior for conformance checking tend to be computationally expensive. Conformance checking techniques need to be scalable with respect to the increasing volume of data, both in terms of the number of observed traces and the size and complexity of process models.

Challenge 2: *Decomposition techniques.* Decomposition techniques have emerged as a promising way of tackling increasing sizes and complexities of processes. By partitioning the process before running conformance checking algorithms on the smaller pieces, decomposition techniques can often achieve substantial performance gains. One family of conformance checking algorithms that decomposition techniques have been applied on

are alignment-based techniques [65, 84, 85, 50, 49]. However, these techniques solves the decisional problem of classifying whether if a given log trace is conforming or yields a lower bound on misalignment costs. The challenge here is being able to apply the divide and conquer paradigm to yield exact conformance results as computed by its monolithic counterpart.

Challenge 3: *Multitudes of algorithms for the same task.* There can be many conformance checking algorithms for the same task, i.e., same input and output. For example, for alignment-based techniques, there are many techniques that focus on being really good at specific aspects, e.g., computation time, specific data type and conformance issues [75, 44, 9, 6, 19]. If the end user is aware of their pros and cons under different circumstances, she would be able to choose the right technique for the job. However, the end user may not have the expert knowledge to always select the most appropriate algorithm. One solution is to have an oracle that have been configured to help users with the decision given their objectives. For example, users may want an algorithm that would quickly process their data, such oracle would help them choose the best algorithm for the job.

Challenge 4: *Online conformance checking.* Given the volume and velocity at which event data comes in, organizations may not store these data for offline analysis and have to resort to online techniques. Moreover, performing analysis in real time allows process stakeholders to react to conformance issues. However, performing conformance checking in an online context pose challenges in various aspects, e.g., computational time and space requirement, warm start scenarios, and uncertainty to how a case may unfold. There are only a few existing online conformance checking works in the literature [15, 16, 78].

1.4. Contributions of the thesis

Table 1.2 summarizes the contributions of this thesis. The table includes a description of the contribution, the chapter in which they are presented, the conformance challenges they address, and their corresponding publications. The remainder of this section gives an overview of the contributions.

Contribution	Chapter	Challenge	Publication
Condition under which sub-alignments computed under decomposition techniques can be merged as an overall alignment with exact misalignment costs	3	1, 2	[38, 37, 34, 36]
Recomposing conformance checking framework that computes fitness with configurable accuracy	4, 5	1, 2	[38, 37, 34, 36]
Application of machine learning to predict the alignment technique that minimizes computation time	6	3	
Online conformance checking technique based on Hidden Markov Model (HMM)	7	4	[35]

Table 1.2. Table summarizing the contributions of this thesis

For many conformance checking techniques, one challenge is that process models often contains parallelism and loop constructs which make the set of possible modeled behavior large and at times infinite. Having to compare observed behavior against such a large set of possible modeled behavior can be computationally expensive even under an offline scenario. To illustrate this, consider the parallel construct shown in Figure 1.3. The number of possible traces $n!$ increases extremely quickly with respect to n, the number of activities. Moreover, if the process model also includes slightly more complex yet common constructs such as a non free choice construct, simpler conformance checking techniques such as comparing the footprint matrices of the model and log would not suffice.

One possible solution is apply a divide-and-conquer approach so that the model is decomposed into fragments. Upon performing conformance checking on the fragments, their conformance results are merged back together as an overall result. To appreciate the effect, suppose that there are 10 activities, i.e., $n = 10$. The number of possible traces equals to $10! = 3628800$. However, if the model could somehow be split into two equal fragments that can be checked separately, then we would only have to compare $5! = 120$

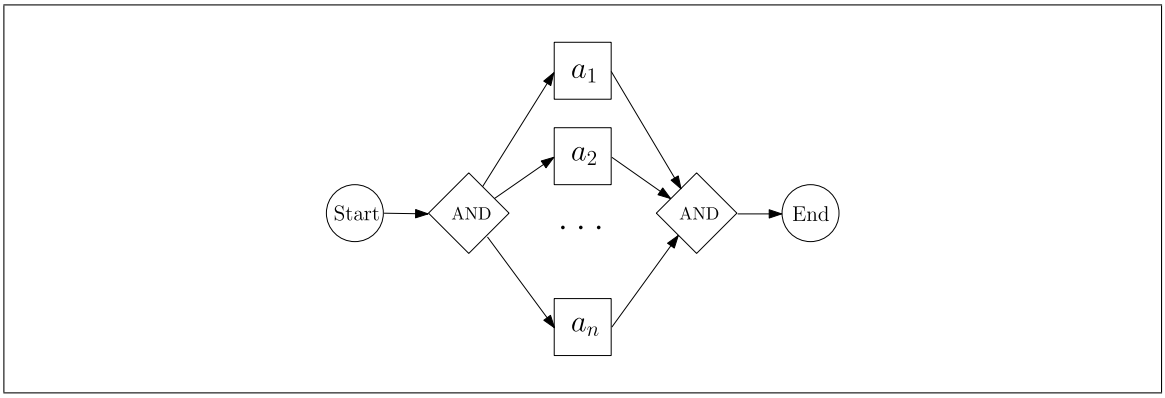


Figure 1.3. Parallel construct with n activities

traces two times, i.e., 240 traces. This is a $15000\times$ difference. Towards this, this thesis contributes:

- Results that show the condition under which alignment results using decomposition techniques can be merged as overall results that have exact conformance values as would have been computed under its monolithic counterpart.
- A novel framework that computes alignment-based fitness using a divide and conquer approach. Furthermore, the framework allows user configuration to adjust the degree of accuracy of the conformance result.
- Extension of the proposed framework to include heuristics that encourages the mergeable condition of decomposed conformance results.

There are many techniques that do the same conformance checking task but each specialize in certain aspects, e.g., finding specific conformance issues, and can have drastically different performance depending on the input data. This means that the end users need expert knowledge to choose the appropriate algorithm depending on their data and objectives. Towards this, the thesis contributes:

- Classifiers that tackle the algorithm selection problem of deciding whether to apply decomposition techniques for a given model and log trace.

Organizations may have to perform conformance checking in online context due to storage limitation, desire to intervene in the occurrence of conformance issues, and other reasons. Online conformance checking brings along several challenges that do not exist in offline conformance checking. Towards this, this thesis contributes:

- An online conformance checking framework that focuses on the problem of balancing between making sense at the process level as the case reaches completion and putting emphasis on the current information at the same time.

1.5. Research objectives

In view of the challenges, the thesis focuses on the following research objectives:

Objective 1: Existing decomposed conformance checking approaches are limited to the decision problem of deciding whether or not an event log is perfectly fitting with a process model. This means that the exact degree of deviation severity, i.e., cost-based fitness, and the exact deviating alignments cannot be yielded. This objective addresses this issue by relaxing the conditions under which results from decomposed subcomponents can be merged back to an overall result.

Objective 2: While it has been shown that decomposition of alignment-based conformance checking can lead to significant performance gains, there is little work on identifying and investigating the different circumstances under which decomposition techniques can contribute significant performance gains. Further understanding of such circumstances and characteristics can enable the appropriate application of different decomposition strategies at different scenarios and drive the design of new decomposition strategies.

Objective 3: Investigate online conformance checking techniques that address the fundamental challenge of explaining the conformance of a running case is in balancing between making sense at the process level as the case reaches completion and putting emphasis on the current information at the same time.

1.6. Hypothesis

This research proposes the following hypothesis: "Given a model and event log, decomposition techniques can be applied for alignment-based conformance checking to yield the overall conformance result." For the online conformance checking problem, the research proposes the following hypothesis: "Given an event stream and model, conformance of the event stream with respect to the model can be computed under the required complexity of stream processing".

1.7. Methodology

To achieve the proposed research goals, the design science research methodology is followed [28] where the produced artifacts are novel approaches that computes the conformance of a model and log in a decomposed manner. The utility and efficacy of the artifacts are demonstrated through mathematical proofs and empirical experiments on both synthetic and real-life datasets.

1.8. Impact

The publications related to this thesis are as follows:

Journal papers

[38] Wai Lam Jonathan Lee, HMW Verbeek, Jorge Munoz-Gama, Wil MP van der Aalst, and Marcos Sepúlveda. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Information Sciences*, 2018

[35] Wai Lam Jonathan Lee, Andrea Burattin, Jorge Munoz-Gama, and Marcos Sepúveda. Orientation and conformance: A HMM-based approach to online conformance checking. *Information System (under review)*, 2019

Conference and poster presentations

[36] Wai Lam Jonathan Lee, Jorge Munoz-Gama, H. M. W. Verbeek, Wil M. P. van der Aalst, and Marcos Sepúlveda. Improving Merging Conditions for Recomposing Conformance Checking. In *Business Process Management Workshops - BPM 2018 International Workshops, Sydney, Australia, September 10, 2018. Revised Papers*, 2018

[34] Wai Lam Jonathan Lee. Advancing Decomposed Conformance Checking in Process Mining. In *Business Process Management Doctoral Consortium, Sydney, Australia, September 09, 2018.*, 2018

[37] Wai Lam Jonathan Lee, H. M. W. Verbeek, Jorge Munoz-Gama, Wil M. P. van der Aalst, and Marcos Sepúlveda. Replay using recomposition: Alignment-based conformance checking in the large. In *Proceedings of the BPM Demo Track and BPM Dissertation Award, Barcelona, Spain, September 13, 2017.*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017

1.9. Document structure

The thesis in structured in four parts:

- Part I presents the introduction that you are currently reading. Also it introduces the necessary preliminaries used throughout the thesis.
- Part II presents our work on extending decomposition techniques to alignment-based conformance checking. Chapter 3 presents the merging condition that allows merged sub-alignments to have the exact misalignment costs as an optimal alignment computed under the monolithic approach. Chapter 4 make use of this property to present an iterative framework that computes optimal alignments in a divide and conquer manner. Chapter 5 presents further extension to the proposed framework to address the bottleneck identified in extensive experimental studies.

- Part III moves to investigate the problem of selecting the best algorithm for the conformance checking job given the input data. Specifically, the chapter applies machine learning to identify scenarios in which decomposition techniques can compute optimal alignments in less time than the monolithic approach.
- Part IV presents our work on online conformance checking where we present a Hidden Markov Model (HMM) based approach that tries to balance between computing conformance results that make sense at the process level as the running case reaches completion and putting emphasis on the current information at the same time.
- Lastly, Part V concludes the thesis, summarizing the main results and discusses possible extensions of the presented work.

2. Preliminaries

In this chapter, we present the preliminaries that will be used later in the thesis. We first recall basic concepts such as sets, multisets, functions, and projections. Then, we present concepts commonly used in process mining such as events, trace, event logs, and process models. We also give a general overview on process mining and an in-depth view on alignment-based conformance checking.

2.1. Basic notations

To facilitate the definition of later concepts, we first introduce some basic notations.

Definition 2.1 (Multisets)**.** *Let X be a set, a multiset of X is a mapping $M : X \to \mathbb{N}$. $\mathcal{B}(X)$ denotes the set of all multisets over X. Let M and M' be multisets over X. M contains M', denoted $M \geq M'$, if and only if $\forall_{x \in X} M(x) \geq M'(x)$. The union of M and M' is denoted $M + M'$, and is defined by $\forall_{x \in X}(M + M')(x) = M(x) + M'(x)$. The difference between M and M' is denoted $M - M'$ and is defined by $\forall_{x \in X}(M - M')(x) = max\ (M(x) - M'(x), 0)$.*

Note that $(M - M') + M' = M$ only holds if $M \geq M'$. For sets X and X' such that $X' \subseteq X$, we consider every set X' to be an element of $\mathcal{B}(X)$, where $\forall_{x \in X'} X'(x) = 1$ and $\forall_{x \in X \setminus X'} X'(x) = 0$.

Definition 2.2 (Projection on sequences and multisets)**.** *Let X be a set, let $X' \subseteq X$ be a subset of X, let $\sigma \in X^*$ be a sequence over X, and let $M \in \mathcal{B}(X)$ be a multiset over X. With $\pi_{X'}(\sigma)$ we denote the projection of σ on X', e.g. $\pi_{\{x,z\}}(\langle x, x, y, y, y, z\rangle) = \langle x, x, z\rangle$. with $\pi_{X'}(M)$ we denote the projection of M on X', e.g. $\pi_{\{x,z\}}([x^2, y^3, z]) = [x^2, z]$.*

Definition 2.3 (Function domains and ranges)**.** *Let $f \in X \nrightarrow X'$ be a (partial) function. With $dom(f) \subseteq X$ we denote the set of elements from X that are mapped onto some value in X' by f. With $rng(f) \subseteq X'$ we denote the set of elements in X' that are mapped onto by some value in X, i.e., $rng(f) = \{f(x) \mid x \in dom(f)\}$.*

Definition 2.4 (Functions on sequences and multisets)**.** *Let $f \in X \nrightarrow X'$ be a (partial) function, let $\sigma \in X^*$ be a sequence of X, and let $M \in \mathcal{B}(X)$ be a multiset of X. With $f(\sigma)$ we denote the application of f on all elements in σ, e.g., if $dom(f) = \{x, z\}$, then $f(\langle x, x, y, y, y, z\rangle = \langle f(x), f(x), f(z)\rangle$. With $f(M)$ we denote the application of f on all elements in M, e.g., if $dom(f) = \{x, z\}$, then $f([x^2, y^3, z]) = [f(x)^2, f(z)]$.*

2.2. Petri nets

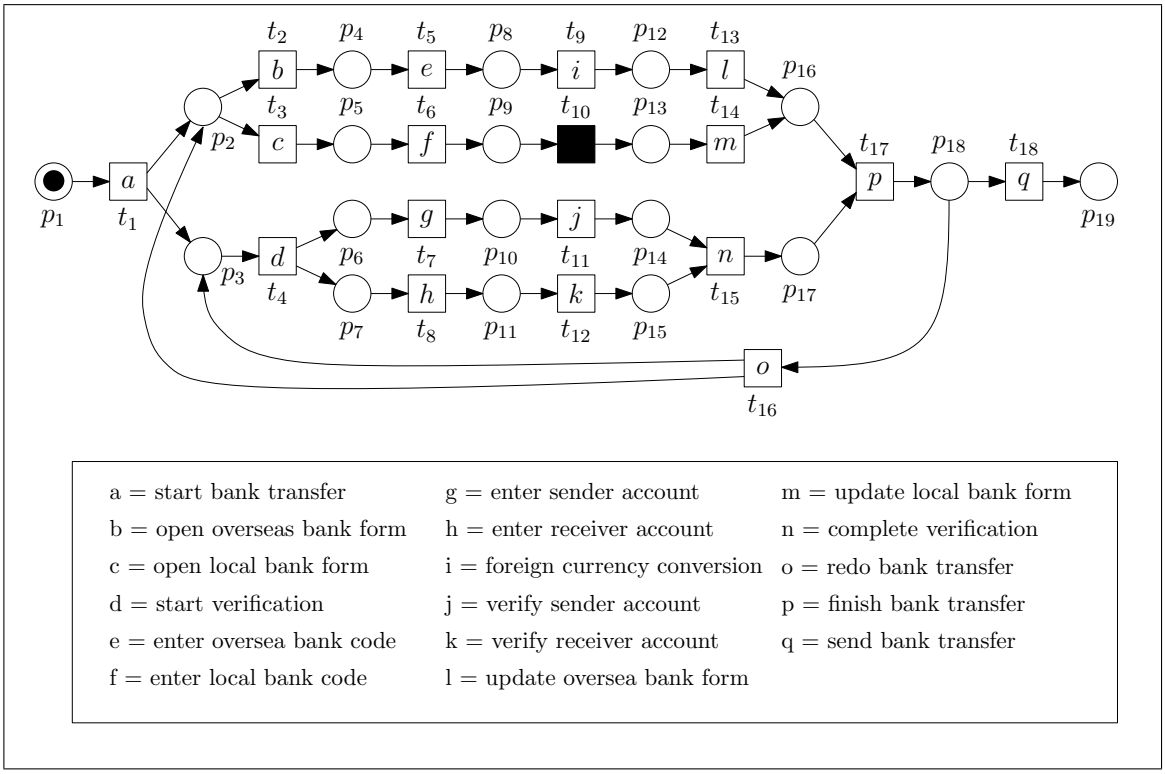


Figure 2.1. Running example: The system net S_1 that contains the (labeled) Petri net N_1

As previously mentioned, processes are depicted using process models. There are many different process modeling languages, e.g., the Business Process Modeling and Notation (BPMN), Event-Driven Process Chains (EPCs), Unified Modeling Language (UML) Activity diagrams, Yet Another Workflow Language (YAWL) and others [24].

In this thesis, we use Petri nets to present our ideas [51], as this is the most often-used process modeling notation in process mining. We stress that process models denoted using Petri nets can be translated to models using other process modeling languages.

Definition 2.5 (Petri net)**.** *A Petri net is a tuple $N = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$ and $F = (P \times T) \cup (T \times P)$ the set of arcs, which is sometimes referred to as the flow relation.*

Places are typically visualized by circles, whereas transitions are typically visualized by squares (or rectangles). Consider the Petri net $N_1 = (P_1, T_1, F_1)$ in Figure 2.1. N_1 has the set of places $P_1 = \{p_1, p_2, \ldots, p_{19}\}$, the set of transitions $T_1 = \{t_1, t_2, \ldots, t_{18}\}$ and the set of arcs $F_1 = \{(p_1, t_1), (t_1, p_2), \ldots, (t_{18}, p_{19})\}$.

The state of a Petri net is called a *marking*, and corresponds to a multiset of places. A marking is typically visualized by putting as many so-called *tokens* (black dots) at a place as the place occurs in the marking. For example, a possible marking of the net N_1 is $[p_2, {p_3}^2]$ which is visualized by one token at place p_2 and two tokens at place p_3.

Definition 2.6 (Marking)**.** *Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e. $M \in \mathcal{B}(P)$*

Let $N = (P, T, F)$ be a Petri net. For a node $n \in P \cup T$ (a place or a transition), $\bullet n = \{n' \mid (n', n) \in F\}$ denotes the set of *input nodes* and $n\bullet = \{n' \mid (n, n') \in F\}$ denotes the set of *output nodes*.

A transition $t \in T$ is *enabled* by a marking M if and only if each of its input places $\bullet t$ contains at least one token in M, that is, if and only if $M \geq \bullet t$. An enabled transition may *fire* by removing one token from each of the input places $\bullet t$ and producing one token at each of the output places $t\bullet$. The firing of an enabled transition t in marking M is denoted as $(N, M)[t\rangle(N, M')$, where $M' = (M - \bullet t) + t\bullet$ is the resulting new marking.

A marking M' is *reachable* from a marking M if and only if there is a sequence of transitions $\sigma = \langle t^1, t^2, \ldots, t^n \rangle \in T^*$ such that $\forall_{0 \leq i < n}(N, M^i)[t^{i+1}\rangle(N, M^{i+1})$ with $M^0 = M$ and $M^n = M'$. For example, $(N_1, [p_1])[\sigma\rangle(N_1, [p_2, p_{17}])$ with the sequence $\sigma = \langle t_1, t_4, t_7, t_8, t_{11}, t_{12}, t_{15} \rangle$ for the Petri net N_1 in Figure 2.1.

Definition 2.7 (Labeled Petri net)**.** *A* labeled *Petri net $N = (P, T, F, l)$ is a Petri net (P, T, F) with labeling function $l \in T \nrightarrow \mathcal{U}_A$ where $\mathcal{U}_A$ is some universe of activity labels. Let $\sigma_v = \langle a^1, a^2, \ldots, a^n \rangle \in \mathcal{U}_A^*$ be a sequence of activities. $(N, M)[\sigma_v \rhd (N, M')$ if and only if there is a sequence $\sigma \in T^*$ such that $(N, M)[\sigma\rangle(N, M')$ and $l(\sigma) = \sigma_v$.*

A transition t is called *invisible* if and only if it is not mapped to any activity label by the labeling function, that is, if and only if $t \notin dom(l)$. Otherwise, transition t is visible and corresponds to an observable activity $a = l(t)$.

Consider the labeled Petri net $N_1 = (P_1, T_1, F_1, l_1)$ in Figure 2.1. It has a labeling function l_1 that maps transition t_1 onto activity label a, t_2 onto b, etc. Note that $t_{10} \notin dom(l_1)$, hence this transition is an invisible transition.

$(N_1, [p_1])[\sigma_v \rhd (N_1, [p_{19}])$ for the sequence $\sigma_v = \langle a, c, f, m, d, g, h, j, k, n, p, q \rangle$ since $(N_1, [p_1])[\sigma\rangle(N_1, [p_{19}])$ with $\sigma = \langle t_1, t_3, t_6, t_{10}, t_{14}, t_4, t_7, t_8, t_{11}, t_{12}, t_{15}, t_{17}, t_{18} \rangle$ and $l_1(\sigma) = \sigma_v$. Note that since t_{10} is not mapped to any activity label, it is not observable in σ_v.

In the context of process mining, the focus is mainly on processes with an initial state and a well-defined final state. For the net N_1, we are interested in *complete* firing sequences, starting from marking $[p_1]$ and ending at marking $[p_{19}]$. The notion of a system net is defined to include the initial and final marking.

Definition 2.8 (System net)**.** *A system net is a triplet $S = (N, I, O)$ where $N = (P, T, F, l)$ is a labeled Petri net, $I \in \mathcal{B}(P)$ is the initial marking and $O \in \mathcal{B}(P)$ is the final marking. $\mathcal{U}_S$ is the* universe of system nets.

The net $S_1 = (N_1, I_1, O_1)$ in Figure 2.1 is a system net, with an initial marking $I_1 = [p_1]$ and a final marking $O_1 = [p_{19}]$. This system net models a bank transfer process in which a client makes a bank transfer from one account (sender account) to another account (receiver account). The receiver account can be of a local bank or an overseas bank.

Consider the visible sequence $\sigma_v = \langle a, b, e, i, l, d, g, h, j, k, n, p, q \rangle$. This sequence describes the activities that are executed for a bank transfer to an overseas bank account. It is initiated with activity a (*start bank transfer*) and is ended with activity q (*send bank transfer*). For an overseas bank transfer, the bank employee has to open a new overseas bank form, enter the overseas bank code, convert the transfer amount into the foreign currency and fill in the bank form with the converted amount. The bank employee also has to verify both the sender and receiver account before making the transfer. The completion of the bank form and the account verification can be done concurrently as shown in S_1. In σ_v, the bank form is completed ($\langle \ldots, b, e, i, l, \ldots \rangle$) before the account verification ($\langle \ldots, d, g, h, j, k, \ldots \rangle$).

Definition 2.9 (System net notations)**.** *Let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$.*

- $T_v(S) = dom(l)$ *is the set of* visible transitions *in* S.
- $A_v(S) = rng(l)$ *is the set of corresponding observable activities in* S.
- $T_v^u(S) = \{t \in T_v(S) \mid \forall_{t' \in T_v(S)} l(t) = l(t') \Rightarrow t = t'\}$ *is the set of unique visible transitions in SN (such that no other transition has the same visible label).*
- $A_v^u(S) = \{l(t) \mid t \in T_v^u(S)\}$ *is the set of corresponding unique observable activities in* S.

For a given system net, the set of visible traces starting from marking I to marking O is projected onto observable activities yields set $\phi(S)$.

Definition 2.10 (Traces)**.** *Let $S = (N, I, O) \in \mathcal{U}_S$ be a system net. $\phi(S) = \{\sigma_v \mid (N, I)[\sigma_v \rhd (N, O)\}$ is the set of visible traces starting in marking I and ending in marking O. $\phi_f(S) = \{\sigma \mid (N, I)[\sigma\rangle(N, O)\}$ is the corresponding set of complete firing sequences.*

For the system net S_1 in Figure 2.1, $\phi(S_1) = \{\langle a, b, e, i, l, d, g, j, h, k, n, p, q\rangle, \langle a, c, f, m, d, g, j, h, k, n, p, q\rangle, \dots\}$ and $\phi_f(S_1) = \{\langle t_1, t_2, t_5, t_9, t_{13}, t_4, t_7, t_{11}, t_8, t_{12}, t_{15}, t_{17}, t_{18}\rangle, \langle t_1, t_3, t_6, t_{10}, t_{14}, t_4, t_7, t_{11}, t_8, t_{12}, t_{15}, t_{17}, t_{18}\rangle, \dots\}$. Due to the loop involving transition t_{16} there is an infinite number of visible traces and complete firing sequences.

The union of two system nets is defined for composing and decomposing of process models.

Definition 2.11 (Union of nets)**.** *Let $S = (N, I, O) \in \mathcal{U}_S$ with $N = (P, T, F, l)$ and $S' = (N', I', O') \in \mathcal{U}_S$ with $N' = (P', T', F', l')$ be two system nets.*

- *$l'' \in (T \cup T') \nrightarrow \mathcal{U}_A$ with $dom(l'') = dom(l) \cup dom(l')$, $l''(t) = l(t)$ if $t \in dom(l)$, and $l''(t) = l'(t)$ if $t \in dom(l') \setminus dom(l)$ is the union of l and l'.*
- *$N \cup N' = (P \cup P', T \cup T', F \cup F', l'')$ is the union of N and N'.*
- *$S \cup S' = (N \cup N', I + I', O + O')$ is the union of systems nets S and S'.*

Note that since the unioned labeling function $l''(t) = l(t)$ rather than $l''(t) = l'(t)$ if $t \in dom(l)$ and $t \in dom(l')$, the union of nets is not commutative, i.e., $S \cup S' \neq S' \cup S$.

2.3. Events, trace, event logs, and event streams

Event logs serve as the starting point for process mining. An event log is a multiset of *traces*. Each trace describes a particular *case*, i.e., a process instance, in terms of the activities executed.

Definition 2.12 (Trace, Event log)**.** *Let $A \subseteq \mathcal{U}_A$ be a set of activities. A trace $\sigma \in A^*$ is a sequence of activities. An event log $L \in \mathcal{B}(A^*)$ is a multiset of traces.*

In this simple definition of an event log, an event refers to just an activity. Often event logs store additional information about events such as resources, timestamps or additional data elements recorded with the event log. In this thesis, we abstract from such information and limit conformance to solely the control flow aspect.

$$
\begin{aligned}
L_1 = [\sigma_1 &= \langle a, b, e, i, l, d, g, j, h, k, n, p, q\rangle^{20}, \\
\sigma_2 &= \langle a, c, f, m, d, g, j, k, h, n, p, q\rangle^{5}, \\
\sigma_3 &= \langle a, e, i, l, d, g, j, h, k, n, p, q\rangle^{5}] \\
L_2 = [\sigma_4 &= \langle a, c, f, m, d, g, j, h, k, n, p, q\rangle^{20}, \\
\sigma_5 &= \langle a, b, e, i, l, d, j, g, h, k, n, p, q\rangle^{5}, \\
\sigma_6 &= \langle a, b, e, i, l, d, g, h, n, j, k, p, q\rangle^{5}]
\end{aligned}
$$

Figure 2.2. Running example: Event logs L_1 and L_2

Consider the event logs L_1 and L_2 in Figure 2.2. Both event logs have three distinct traces. L_1 contains 30 traces, with 20 σ_1 traces, 5 σ_2 traces and 5 σ_3 traces. L_2 contains 30 traces with 20 σ_4 traces, 5 σ_5 traces and 5 σ_6 traces. Note that an event log is a multiset of traces as a distinct trace (like σ_1) can occur multiple times.

The projection function $\restriction_X$ as introduced earlier also directly applies to event logs. For example, consider the event log L_1 in Figure 2.1, $\pi_{\{a,b,e\}}(L_1) = [\langle a, b, e\rangle^{20}, \langle a\rangle^5, \langle a, e\rangle^5]$ and $\pi_{\{a,b\}}(L_1) = [\langle a, b\rangle^{20}, \langle a\rangle^{10}]$. For a log $L_3 = [\langle\rangle]$ with an empty trace, the projection $\pi_{\{a,b,e\}}(L_3) = [\langle\rangle]$ returns a log with an empty trace. These projected event logs are referred to as sublogs and the traces in sublogs are referred to as subtraces.

While process mining is often done offline on event data of completed process executions, it is desirable perform online analysis on streams of incoming event data on ongoing process executions. Similar to [16], we conceptualize event streams as streams of observable units where each observable unit contains process information of an event.

Definition 2.13 (Observable unit)**.** *Let $C \subseteq \mathcal{U}_C$ denote the set of case ids, and let $A \subseteq \mathcal{U}_A$ denote the set of activities. An observable unit $o = (c, a) \in C \times A$ is a pair describing an activity a observed in context of case id c. The universe of all possible observable units is defined as $O = C \times A$.*

The activities of observable units correspond to observations of fired transitions in the corresponding Petri net model. Projection operators can be used to extract the case id and

the activity, i.e., given $o = (c, a)$, $\pi_c(o) = c$ and $\pi_a(o) = a$. Moreover, an event stream is simply an infinite sequence of observable units.

Definition 2.14 (Event stream)**.** *Given the universe of observable units $O = C \times A$, an event stream is defined as an infinite sequence of observable units: $S : \mathbb{N}_{\geq 0} \to O$.*

As such, an event stream can be seen as an unbounded sequence of observable units for which the sequence order corresponds to the chronological order of the corresponding events. Where it is clear that we are referring to an event stream of only one case, we will directly refer to the corresponding activities rather than apply the projection operators for each observable unit.

2.4. Process mining

In Section 2.2 and Section 2.3, we have presented various concepts on process models and event data. In this thesis, we propose techniques that improve conformance checking in process mining. As such, we recall two common problems in process mining, namely process discovery and conformance checking to provide context for the rest of the thesis.

2.5. Process discovery

Process discovery involves discovering a process model from an event log with the purpose of helping users to understand various aspects of the underlying process. There are many process discovery algorithms [71, 41, 40, 81]. However, in most cases, the discovered model might not encompass all the observed behavior in the event log or might allow behavior unseen in the observed behavior. This motivates conformance checking for identifying commonalities and discrepancies between the observed behavior in the event log and the modeled behavior in the discovered process model.

2.6. Alignment-based conformance checking

The main idea of conformance checking is to compare the observed behavior of an event log $L \in \mathcal{B}(A^*)$ with the modeled behavior of the related model, i.e. the related system net $S = (N, I, O)$.

Similar to process discovery, many techniques have been proposed for conformance checking [1, 6, 18, 92, 58]. In this thesis, we mainly focus on alignment-based techniques due to their robustness and level of detail on analysis.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ \hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & c & f & \gg & m & d & g & j & k & h & \gg & n & p & q \\ \hline a & c & f & \tau & m & d & g & j & \gg & h & k & n & p & q \\ t_1 & t_3 & t_6 & t_{10} & t_{14} & t_4 & t_7 & t_{11} & & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & \gg & e & i & l & d & g & j & h & k & n & p & q \\ \hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

Figure 2.3. Alignments for event log L_1

Alignment-based techniques mostly focus on the fitness metric to identify discrepancies between the model and event log. As hinted by the name, these techniques compute alignments between the traces in the event log and the visible traces of the process model. Consider the alignments of the three traces in the event log L_1 in Figure 2.3. For each alignment, the top row corresponds to the trace in the event log, the middle row corresponds to the visible trace in the model, whereas the bottom row corresponds to the corresponding firing sequence in the model.

If an activity in the model cannot be mimicked by an activity in the log, then a $\gg$ ("no step") appears in the top row. Similarly, if an activity in the log cannot be mimicked by an activity in the model, then a $\gg$ ("no step") appears in the bottom row. Note that we use the symbol τ as the surrogate activity label for invisible transitions.

For example, the fourth column of γ_2 indicates that the net S_1 has fired transition t_{10}, which is an invisible transition, and that the log trace σ_2 could not mimic this firing. The ninth column indicates that the activity k was performed in the log trace σ_2, but that the net S_1 could not mimic this. These columns containing $\gg$ point to deviations between model and log.

A *move* is a pair (a, m) where the first element a refers to the activity in the log and the second element m refers to the transition in the net.

Definition 2.15 (Legal moves)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$. $A_M = \{(a, (a, t)) \mid a \in A \wedge t \in T \wedge l(t) = a\} \cup \{(\gg, (a, t)) \mid a \in A \wedge t \in T \wedge l(t) = a\} \cup \{(\gg, (\tau, t)) \mid t \in T \wedge t \notin dom(l)\} \cup \{(a, \gg) \mid a \in A\}$ is the set of* legal moves*. The function $\alpha \in A_M \to A \cup \{\tau\}$ provides the activity (possibly τ) associated with a move: for all $t \in T$ and $a \in A$, $\alpha(a, (a, t)) = a$, $\alpha(\gg, (a, t)) = a$, $\alpha(\gg, (\tau, t)) = \tau$, and $\alpha(a, \gg) = a$.*

An alignment is a sequence of legal moves. This means that, after removing all $\gg$ symbols, the top row corresponds to a log trace and the bottom row corresponds to a firing sequence in the net from marking I to marking O. The middle row corresponds to a visible trace after also removing all τ symbols.

Definition 2.16 (Alignment)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$, let $\sigma_L \in L$ be a log trace and $\sigma_M \in \phi_f(S)$ a complete firing sequence of system net S. An* alignment *of σ_L and σ_M is a sequence $\gamma \in A_M^*$ such that the projection on the first element (ignoring any $\gg$) yields σ_L and the projection on the last element (ignoring any $\gg$) yields σ_M.*

$$\gamma_2' = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & c & f & \gg & m & d & g & j & k & h & \gg & \gg & \gg & n & p & q \\ \hline a & c & f & \tau & m & d & \gg & \gg & \gg & h & k & g & j & n & p & q \\ t_1 & t_3 & t_6 & t_{10} & t_{14} & t_4 & & & & t_8 & t_{12} & t_7 & t_{11} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

Figure 2.4. Alternative alignment for trace σ_2

Given a log trace and a model, there could be multiple or even infinitely many alignments. Consider trace σ_2 in event log L_1 in Figure 2.2. One possible alignment is γ_2 at Figure 2.3 while another could be γ_2' at Figure 2.4. It is clear that γ_2 is a better alignment as it better matches the log trace with the model trace. In general, costs can be assigned to different types of moves so that an optimal alignment with the lowest costs can be computed.

Definition 2.17 (Cost of alignment)**.** *The cost function $\delta \in A_M \to \mathbb{Q}$ assigns costs to legal moves. Moves where the log and the model agree have no costs, i.e. $\delta(a,(a,t)) = 0$ for all $a \in A$. A move in the model also has no costs if the transition is invisible, i.e. $\delta(\gg,(\tau,t)) = 0$ if $t \notin dom(l)$. A move in the model has a cost of $\delta(\gg,(a,t)) > 0$ if $l(t) = a$ and $a \in A$. Similarly, a move in the log has a cost of $\delta(a,\gg) > 0$. The* cost *of an alignment $\gamma \in A_M^*$ is the sum of all costs: $\delta(\gamma) = \sum_{(a,m)\in\gamma} \delta(a,m)$.*

In this paper, we assume a standard cost function δ_1 that assigns unit costs: $\delta_1(a,(a,t)) = 0$, $\delta_1(\gg,(\tau,t)) = 0$ and $\delta_1(\gg,(a,t)) = \delta_1(a,(\gg,t)) = 1$ for all $a \in A$. For example, $\delta_1(\gamma_1) = \delta_1(\gamma_4) = 0$, $\delta_1(\gamma_2) = \delta_1(\gamma_5) = \delta_1(\gamma_6) = 2$ and $\delta_1(\gamma_3) = 1$.

Definition 2.18 (Optimal alignment)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $S \in \mathcal{U}_S$ be a system net with $\phi(S) \neq \emptyset$.*

- *For $\sigma_L \in L$, an alignment γ between σ_L and a complete firing sequence of the system net $\sigma_M \in \phi_f(S)$ is optimal if the associated misalignment costs are lower or equal to the costs of any other possible alignment γ'.*

- $\lambda(\sigma_L, S) \in A^* \rightarrow A_M^*$ *is a deterministic mapping that assigns any log trace* σ_L *to an optimal alignment.*

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & c & f & \gg & m & d & g & j & h & k & n & p & q \\ \hline a & c & f & \tau & m & d & g & j & h & k & n & p & q \\ t_1 & t_3 & t_6 & t_{10} & t_{14} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_5 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & e & i & l & d & j & g & \gg & h & k & n & p & q \\ \hline a & b & e & i & l & d & \gg & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_6 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & e & i & l & d & g & h & n & j & k & \gg & p & q \\ \hline a & b & e & i & l & d & g & h & \gg & j & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_8 & & t_{11} & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

Figure 2.5. Alignments for event log L_2

Consider the system net S_1 in Figure 2.1 and the event logs L_1 and L_2 in Figure 2.2. The alignments γ_1, γ_2, and γ_3 in Figure 2.3 are possible optimal alignments for the traces in L_1 and their corresponding firing sequences in the system net S_1. The alignments γ_4, γ_5, and γ_6 in Figure 2.5 are possible optimal alignments for the traces in L_2 and their corresponding firing sequences in the same system net. The mapping function would return one of the possible optimal alignments in a deterministic manner so that the same one is always returned.

With an optimal alignment γ between σ_L and S, we can quantify fitness by comparing its associated misalignment costs with the costs of a default alignment.

Definition 2.19 (Fitness metric)**.** *Let* $L \in \mathcal{B}(A^*)$ *be an event log and let* $S = (N, I, O) \in \mathcal{U}_S$ *be a system net with* $N = (P, T, F, l)$*. Let* $l^\tau \in T \rightarrow A$ *be such that* $l^\tau(t) = l(t)$ *if* $t \in dom(l)$ *and* $l^\tau(t) = \tau$ *if* $t \notin dom(l)$*. Let* $\sigma_L \in L$ *be a log trace. Let*

$move_M(S) = min_{\sigma_M \in \phi_f(S)} \sum_{t \in \sigma_M} \delta(\gg, (l^\tau(t), t))$ *be the minimal costs of an alignment between an empty log trace and a complete firing sequence of the system net. Let* $move_L(\sigma_L) = \sum_{a \in \sigma_L} \delta(a, \gg)$ *be the costs of an alignment between* σ_L *and an empty model trace.*

Given the single-trace fitness function fit*, the fitness of the trace* σ_L *is computed as follows:*

$$\mathit{fit}(\sigma_L, S, \delta) = 1 - \frac{\delta(\lambda(\sigma_L, S))}{move_M(S) + move_L(\sigma_L)}$$

The fitness function is also overloaded to compute the fitness of the event log L *as follows:*

$$\mathit{fit}(L, S, \delta) = 1 - \frac{\sum_{\sigma_L \in L} \delta(\lambda(\sigma_L, S))}{|L| \times move_M(S) + \sum_{\sigma_L \in L} move_L(\sigma_L)}$$

This is a relative fitness metric presented in many conformance related papers [5, 4, 67]. The metric normalizes the misalignment costs associated to γ by the costs of the extreme case, a default alignment where all the steps in the log trace are aligned as log moves and all the steps of the minimum model trace are aligned as model moves. Using an optimal alignment $\gamma = \lambda(\sigma_L, S)$, the fitness metric computes a value between 0 and 1. A trace that perfectly fits the system net would yield a fitness value of 1 and a trace that does not fit the system net at all would yield a fitness value of 0.

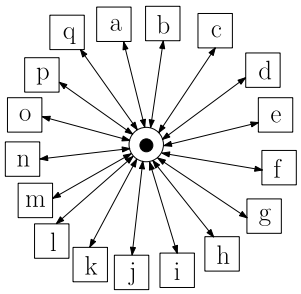
Consider the trace σ_3 in Figure 2.2 and the net S_1 in Figure 2.1. Assuming that the optimal alignment $\gamma_3 = \lambda(\sigma_3, S_1)$ in Figure 2.3 is used, the fitness of σ_3 and S_1 is $\mathit{fit}(\sigma_3, S_1, \delta_1) = 1 - \frac{1}{12+12} = \frac{23}{24} \approx 0.958$. The fitness of the event log L_1 and S_1 is $\mathit{fit}(L_1, S_1, \delta_1) = 1 - \frac{0 \times 20 + 2 \times 5 + 1 \times 5}{12 \times 30 + 13 \times 20 + 12 \times 5 + 12 \times 5} = 1 - \frac{15}{740} = \frac{145}{148} \approx 0.980$. Recall that the cost function δ as a function is defined for multisets. This means that the cost function δ is applied to a trace σ multiple times if there are multiple cases with the trace σ.

2.7. Beyond fitness

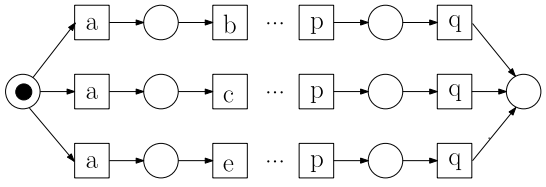
While fitness is often an important concern for process stakeholders, there are other metrics that one should attend to. In fact, there are four commonly accepted quality dimensions for comparing a model and a log: (1) fitness, (2) precision, (3) simplicity and (4) generalization [56].

As previously presented, a model has good fitness if it can mimic the behavior of the event log. Yet a fitting model is not necessarily a good model. For example, Figure 2.6a shows a "flower model" which is able to replay all the traces in the event log L_1 and L_2 and allows sequences that are not seen in the event log. Such a model does not contain any knowledge of the process other than its activities and is unlikely to be of much use. A model is precise if it does not allow "too much" behavior. A model that lacks precision is underfitting.

In contrary, Figure 2.6b shows a model that does not allow any behaviour other than the traces in the event log L_1. While it is perfectly fitting and precise with respect to log L_1, it is also unlikely to be a good model. An event log often does not contain all the possible runs of the process. Generalization means that a model should not be overfitting. Lastly, the simplicity dimension refers to Occam's Razor; a model should be as simple as possible.



(a) Imprecise flower model that yields perfect fitness



(b) Non-generalizing model that only allow observed traces in event log L_1

Similar to the fitness metric, different techniques have been proposed to measure precision, generalization, and simplicity. For example, the escaping arc precision metric computes precision quickly. It first performs a log traversal over the model to compute a *prefix automaton* where there is a state for each unique prefix of the event log. Then it measures the proportion of enabled transitions that are unseen in the observed log behavior as "escaping arcs" [47]. Computing the escaping arc precision is fast since it only looks at the markings reached by replaying log traces over the model instead of exploring the entire state-space of the model. In the case where there are non-fitting log traces that cannot be replayed onto the model, one can use the model projection of an optimal alignment between the log trace and model as the corresponding replayable trace. This is known as the one-alignment approach [47]. However, as shown in [60], there are several limitations of the escaping arc precision metric. For example, since there can be multiple optimal alignments between a log trace and model, the escaping arc precision becomes non-deterministic which is certainly undesirable (imagine a very puzzled process analyst)!

Another precision metric is the Markovian Abstraction Precision (MAP) [7]. This metric works by abstracting both the model and log into a so-called k^{th}-order Markovian abstraction (M^k-abstraction) for comparison. A M^k abstraction is a graph where each node is a sub-trace of at most length k and each edge connects nodes with overlapping and consecutive sub-traces. After constructing the M^k-abstractions for both the log and model, precisio1n is computed with respect to the best matching of edges of the model M^k-abstraction to the log M^k-abstraction. By having a configurable maximum sub-trace length, this metric is able to tackle the problem of potentially having to check an exponential number of reachable points in the model for unobserved yet permitted behavior. Furthermore, it was shown that the metric fulfills the proposed five precision axioms [60] for some k. However, as reported by the authors, the needed k for computing accurate precision may be large. This can make both the construction and edge matching computationally expensive.

Yet another precision metric can be derived from the notion of anti-alignments [18, 76]. An anti-alignment is a model trace that *most deviates* from all of the observed traces in the log. The idea here is that if a model is precise with respect to the log, then its anti-alignments will resemble closely to the observed traces. To capture the notion of "most deviation", Hamming distance is used, and truncation and padding are adopted to enable the comparison of traces with different lengths. As such, an anti-alignment is defined with parameters (n, m) where n corresponds to the length of the anti-alignment and m corresponds to the minimum distance between the anti-alignment and all the log traces. Anti-alignments can be computed as SAT problems. Since the number of mismatches between an (n, m)-anti-alignment is bounded within $[1, n]$, it reflects how dissimilar the model is with respect to the observed behavior. Moreover, it can be converted into a precision metric that monotonically increases with log size. The use of anti-alignments to compute conformance metrics was further investigated in [76]. In that work, the authors proposed that precision can be computed by computing anti-alignments at both the trace and log level. At the trace level, the idea is that if the model is precise then the anti-alignment of the model with respect to the log excluding a single trace would correspond to the excluded log trace. One can see that with the overly precise model in Figure 2.6b where if one would exclude a single trace in log L_1 and compute the anti-alignment between the model and log, the excluded log trace would be the model trace that differs most with respect to the modified log. Then the trace-based precision is computed using the distance between the anti-alignment and the excluded log trace. At the log level, the anti-alignment is computed against the entire log and the log-based precision is computed by comparing the anti-alignment and the log. The paper also extends to the generalization quality dimension that we have yet to present much about. As recalled, generalization is about the model permitting unobserved behavior so that it has some flexibility. In the paper, the concept of recovery distance was proposed. Recovery distance refers to the maximum distance between any of the states reached in the anti-alignment and the states visited by the log [76]. Then, the idea is that a generalizing model introduces new behavior but not new states. This is conceptualized as a generalization metric that favors high

anti-alignment distance and low recovery distance. Similar to the precision metric, generalization can be computed at both the trace and log level. Similar to cost-based alignment, computing anti-alignment can be computationally expensive, specifically it was shown to be NP-complete.

We have now presented several existing conformance metrics that measure different conformance dimensions. We emphasize that there are many other metrics that we have not talked about, e.g., advanced behavioral appropriateness [57], projected conformance checking [42], negative events precision [83] and many more. However, while many of these metrics work on small, often toy, examples, they can have difficulty scaling to industrial-sized processes that one might find in real settings. This marks the theme of this thesis where we explore different ways of enabling conformance checking under different contexts. One principal avenue of research was on decomposition techniques for computing cost-based alignments.

Part II

A divide and conquer approach to alignment

3. Merging decomposed sub-alignments

3.1. Introduction

As of current, alignment-based techniques is the state of the art for measuring fitness between a given event log and model. However, alignments are computationally expensive as the algorithm has to explore a large amount of states to yield an optimal alignment that provides a "best" explanation of the discrepancies between the observed and modeled behavior. Decomposition techniques have emerged as a promising approach to reduce the computational complexity. Rather than aligning the overall event log and model, decomposition techniques first partition them into a set of sub-models and sub-logs so that the alignment procedure is performed on these smaller sub-components [65]. If a deviation is found at one of the subcomponents, then it is clear that there is a deviation in the overall component. Otherwise, the overall process and the overall log are perfectly fitting. As such, alignment problems can be decomposed and distributed over a network of computers. Experimental results based on large-scale applications of decomposition techniques have shown significant improvements in performance, especially in computation time [50].

Following this idea, existing decomposition techniques have tackled the original problem in two different ways. One is the computation of conformance at the decomposed subcomponents level [50, 91, 65], where instead of solving the overall problem, it focuses on identifying local conformance issues at individual subcomponents. The other is the approximation of the overall conformance between the process and the log, such as pseudo-alignments [88] or approximate alignments [61]. As such, it is clear that current decomposition techniques do not address the problem of computing the exact fitness between a given log and model.

This chapter presents results that show the required condition for merging optimal decomposed pseudo-alignments into optimal valid alignments. The merged alignments are

then shown to have the same costs as the alignments computed under the monolithic approach. This approach creates a full circle approach for decomposed alignment ('there and back again'), and makes it possible to solve alignment-based conformance problems that current techniques cannot handle. The remainder of this chapter is organized as follows: Section 3.2 presents valid decompositions and decomposed alignment-based conformance checking. Section 3.4 presents the condition under which merged sub-alignments have the optimal misalignment cost and exact overall fitness.

3.2. Decomposed alignment

Alignment-based conformance checking can be time consuming. This is because the time needed for computing conformance and optimal alignments is heavily influenced by the size of the net and the log, as well as by the complexity of the underlying process. One of the ways to tackle this limitation is through decomposition techniques. The alignment problem can be decomposed by splitting the overall net and the overall log into subcomponents (subnets and sublogs) and then solving this set of smaller problems. Under the assumption that the complexity of the alignment algorithm is significantly worse than linear, solving multiple small alignment problems is often faster than solving one large alignment problem. In addition, the set of decomposed alignment problems can be distributed over a network of computer nodes to further reduce computation time.

However, existing decomposition techniques have limited applicability in computing the overall conformance between the net and the log at the alignment level, i.e., computing conformance using optimal alignments between the net and log traces. Following the decomposition of the overall net and the overall log, existing decomposition techniques only guarantee that the aggregation of conformance results from subcomponents will reflect the exact overall conformance if there is perfect fitness between the net and the log [65]. This has led to the focus on using decomposition to identify problematic sections of the process.

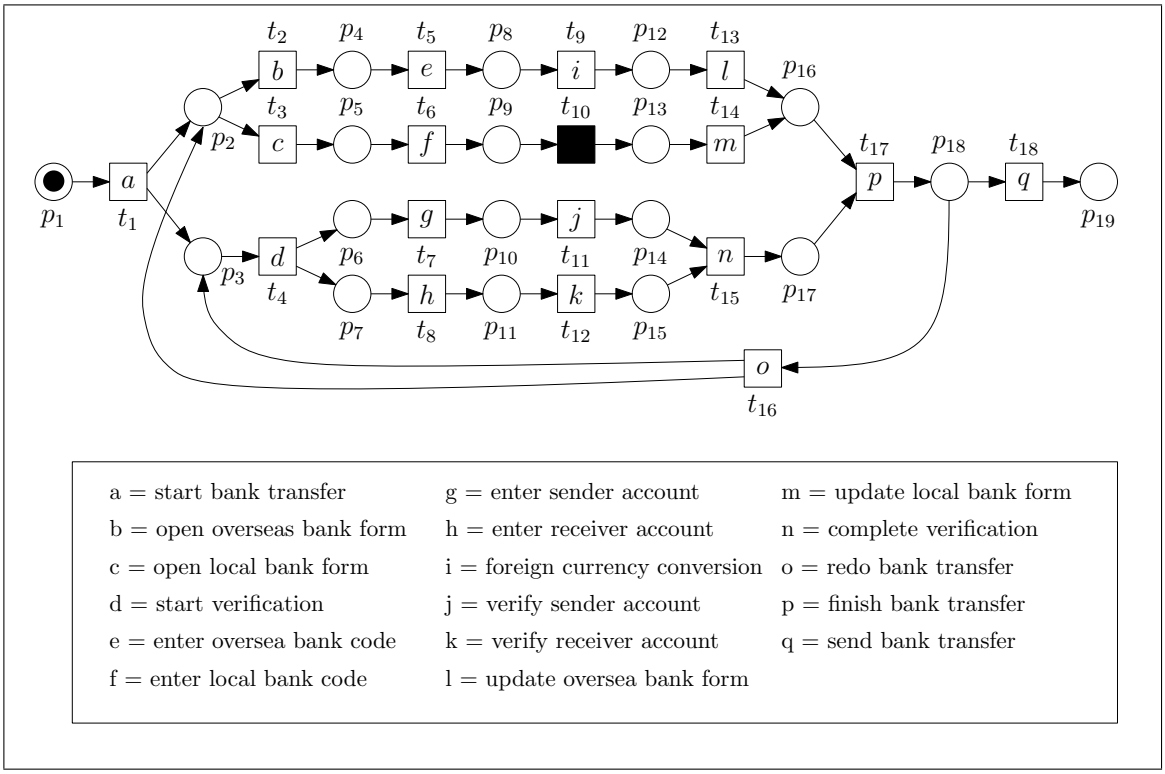


Figure 3.1. Running example: The system net S_1 that contains the (labeled) Petri net N_1

As previously mentioned, there are many scenarios where the precise alignments or costs are required. This motivates our work on extending the conditions under which the conformance results from subcomponents can be aggregated to reflect the exact overall conformance. In this section, we present the core ideas that will extend the applicability of decomposition techniques in computing the overall fitness. These ideas will be later used in two novel alignment-based conformance checking methods to compute an exact or an interval overall fitness result.

3.3. Running example

We first recall the running examples of system net S_1 and logs L_1 and L_2 from Chapter 2.7.

$$L_1 = [\sigma_1 = \langle a, b, e, i, l, d, g, j, h, k, n, p, q\rangle^{20},$$
$$\sigma_2 = \langle a, c, f, m, d, g, j, k, h, n, p, q\rangle^{5},$$
$$\sigma_3 = \langle a, e, i, l, d, g, j, h, k, n, p, q\rangle^{5}]$$

$$L_2 = [\sigma_4 = \langle a, c, f, m, d, g, j, h, k, n, p, q\rangle^{20},$$
$$\sigma_5 = \langle a, b, e, i, l, d, j, g, h, k, n, p, q\rangle^{5},$$
$$\sigma_6 = \langle a, b, e, i, l, d, g, h, n, j, k, p, q\rangle^{5}]$$

Figure 3.2. Running example: Event logs L_1 and L_2

Next, we define valid decompositions of Petri nets and show how alignments computed using valid decompositions can be used to bound the exact fitness value within an interval.

3.3.1. Border activities

In [65] the author presented the concept of *valid decomposition* of a Petri net. A decomposition of a Petri net is valid if each place and invisible transition resides in just one subnet. Moreover, if there are multiple transitions with the same label, they should reside in the same subnet. Only unique visible transitions can be shared among different subnets.

Definition 3.1 (Valid decomposition [65])**.** *Let $S \in \mathcal{U}_{SN}$ be a system net with labeling function l. $D = \{S^1, S^2, \ldots, S^n\} \subseteq \mathcal{U}_S$ is a valid decomposition if and only if the following properties are fulfilled:*

- *$S^i = (N^i, I^i, O^i)$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ for all $1 \leq i \leq n$.*
- *$l^i = \pi_{T^i}(l)$ for all $1 \leq i \leq n$.*
- *$P^i \cap P^j = \emptyset$ for all $1 \leq i < j \leq n$.*

- $T^i \cap T^j \subseteq T^u_v(S)$ *for all* $1 \leq i < j \leq n$.
- $S = \bigcup_{1 \leq i \leq n} S^i$.

$\mathcal{D}(S)$ *is the set of all valid decompositions of* S.

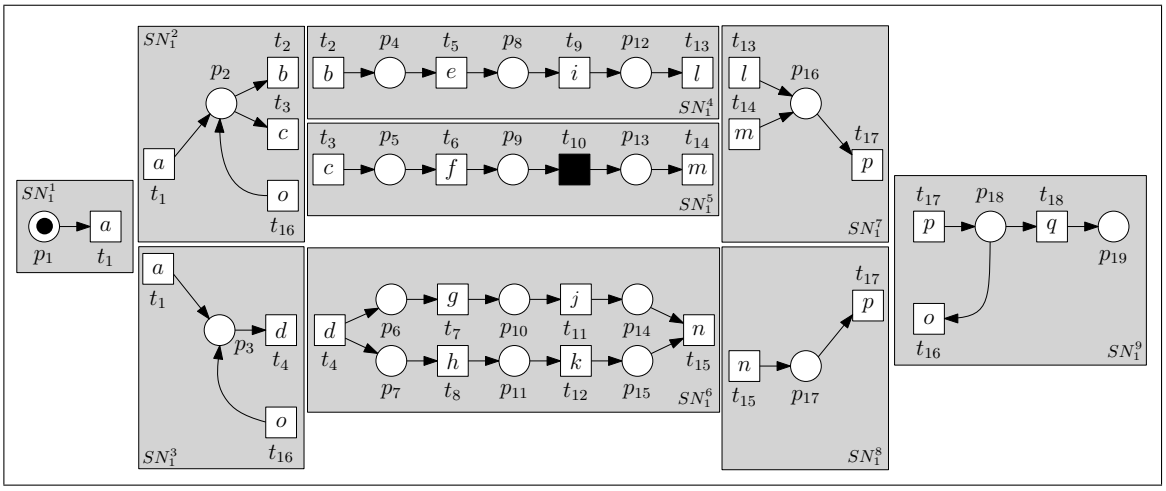


Figure 3.3. Components resulting from a possible valid decomposition D_1 of the system net S_1

For example, the decomposition D_1 in Figure 3.3 is a valid decomposition of the system net S_1, that is, $D_1 \in \mathcal{D}(S_1)$. The system nets in D_1 are referred to as the subnets of S_1.

Given a valid decomposition, an activity may be shared by multiple subnets. We define these activities as the *border activities* of the decomposition. This overlapping property will be used later as a common ground between subcomponents in order to obtain an overall result from local results.

Definition 3.2 (Border activities)**.** *Let* $S = (N, I, O) \in \mathcal{U}_S$ *be a system net with* $N = (P, T, F, l)$. *Let* $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ *be a valid decomposition of* S. *For all* $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i,)$ *is a subnet with* $N^i = (P^i, T^i, F^i, l^i)$. $A_b(D) = \{l(t) \mid \exists_{1 \leq i < j \leq n}\, t \in T^i \cap T^j\}$ *is the set of border activities of the valid decomposition* D.

For an activity $a \in rng(l)$, $S_b(a, D) = \{S^i \mid S^i \in D \wedge a \in A_v(S^i)\}$ is the set of subnets that contain a as an observable activity.

Due to the properties of a valid decomposition, a border activity can only be an activity that has a unique label, i.e., $A_b(D) \subseteq A_v^u(S)$. For example, the valid decomposition D_1 in Figure 3.3 has the set of border activities of $\{a, b, c, d, l, m, n, o, p\}$.

In addition, non-unique activities will appear in precisely one subnet, i.e., for all $a \in A_v(S) \setminus A_v^u(S)$ it holds that $|S_b(a, D)| = 1$. Contrastingly, unique activities may appear in multiple subnets, i.e., for all $a \in A_v^u(S)$ it holds that $|S_b(a, D)| \geq 1$. Border activities are unique activities that appear in multiple subnets, i.e., for all $a \in A_b(D)$ it holds that $|S_b(a, D)| > 1$.

For example with the valid decomposition D_1 in Figure 3.3, $S_b(a, D_1) = \{S_1^1, S_1^2, S_1^3\}$ as the border activity a is shared by the subnets S_1^1, S_1^2 and S_1^3.

3.3.2. Alignment for subnets with border activities

Following a valid decomposition, it can be the case where border activities would either have no input places or output places in the subnets which share the border activities but do not contain the corresponding places. This means that these subnets would have an empty initial marking and/or an empty final marking.

For example, in the valid decomposition D_1 in Figure 3.3, the activity set of S_1^8 consists of the border activities n and p. For subnet S_1^8, border activity n has no input places and border activity p has no output places. This means subnet S_1^8 has an empty initial and final marking. We note that the computation of an optimal alignment remains the same as for system nets with non-empty initial and final markings.

For the sake of simplicity, say we are aligning the subtrace $\sigma_3^8 = \pi_{A_1^8}(\sigma_3) = \langle n, p \rangle$ with subnet S_1^8, i.e., subtrace σ_3^8 is obtained by projecting trace σ_3 onto the activity set of subnet S_1^8. At the start of the alignment procedure, we would yield an empty alignment

since no legal moves have been added yet. Since the initial marking equals the final marking (both are empty), by Definition 2.10, an empty sequence is a valid complete firing sequence. However, the empty alignment is not a valid alignment in this case since Definition 2.16 requires the projection on the first element (ignoring any $\gg$) to yield the log trace, i.e., σ_3^8, and the projection on the last element (ignoring any $\gg$) to yield a complete firing sequence of the net. While the projection on the last element of an empty alignment gives a complete firing sequence, the projection on the first element does not yield σ_3^8. As such, we try to fire the corresponding transition of activity n from subtrace σ_3^8. Transition t_{15} is always enabled in subnet S_3^8 since it has no input places. Firing transition t_{15} produces a token in place p_{17} and enables transition t_{17}. This means that the log and model execution of activity n corresponds to a synchronous move. Similarly, the corresponding transition of activity p, which is the next activity in subtrace σ_3^8, can be fired in the model as there is a token in its input place p_{17}. This means that the log and model execution of activity p corresponds to a synchronous move. Since transition t_{17} has no output places, the marking reached by its firing is an empty multiset. This corresponds to the final marking and is therefore a complete firing sequence. The produced alignment corresponds to subalignment γ_3^8 in Figure 3.4. This is a valid alignment as the projection on the first element (ignoring any $\gg$) yields σ_3^8 and the projection on the last element (ignoring any $\gg$) yields a complete firing sequence.

We note that there are alternative decomposed replay approaches that do not involve removing all irrelevant transitions, places, and arcs. The paper [85] presents the *Hide* and *Hide and Reduce* decomposed replay approaches which create subnets by making irrelevant transitions invisible. Lastly, we refer interested readers to the paper [65] for further details on the replay of subnets created by valid decompositions.

3.3.3. Decomposed Fitness

Let us consider trace $\sigma_3 = \langle a, e, i, l, d, g, j, h, k, n, p, q \rangle$ and the system net S_1 in Figure 3.1. Under a decomposition, the optimal subalignments $\gamma_3^1, \ldots, \gamma_3^9$ can be obtained

$$\gamma_3^1 = \begin{array}{|c|}\hline a \\ \hline a \\ t_1 \\ \hline \end{array} \quad \gamma_3^2 = \begin{array}{|c|c|}\hline a & \gg \\ \hline a & b \\ t_1 & t_2 \\ \hline \end{array} \quad \gamma_3^3 = \begin{array}{|c|c|}\hline a & d \\ \hline a & d \\ t_1 & t_4 \\ \hline \end{array} \quad \gamma_3^4 = \begin{array}{|c|c|c|c|}\hline \gg & e & i & l \\ \hline b & e & i & l \\ t_2 & t_5 & t_9 & t_{13} \\ \hline \end{array} \quad \gamma_3^5 = \begin{array}{|c|}\hline \\ \hline \\ \\ \hline \end{array}$$

$$\gamma_3^6 = \begin{array}{|c|c|c|c|c|c|}\hline d & g & j & h & k & n \\ \hline d & g & j & h & k & n \\ t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} \\ \hline \end{array} \quad \gamma_3^7 = \begin{array}{|c|c|}\hline l & p \\ \hline l & p \\ t_{13} & t_{17} \\ \hline \end{array} \quad \gamma_3^8 = \begin{array}{|c|c|}\hline n & p \\ \hline n & p \\ t_{15} & t_{17} \\ \hline \end{array} \quad \gamma_3^9 = \begin{array}{|c|c|}\hline p & q \\ \hline p & q \\ t_{17} & t_{18} \\ \hline \end{array}$$

Figure 3.4. Subalignments between the trace $\sigma_3 = \langle a, e, i, l, d, g, j, h, k, n, p, q \rangle$ in event log L_1 and the valid decomposition D_1 in Figure 3.3

by first projecting σ_3 onto the subnets $S_1^1, \ldots, S_1^9$ in Figure 3.3 and later aligning each subtrace with the corresponding subnet as shown in Figure 3.4.

$$\gamma_1^1 = \begin{array}{|c|}\hline a \\ \hline a \\ t_1 \\ \hline \end{array} \quad \gamma_1^2 = \begin{array}{|c|c|}\hline a & b \\ \hline a & b \\ t_1 & t_2 \\ \hline \end{array} \quad \gamma_1^3 = \begin{array}{|c|c|}\hline a & d \\ \hline a & d \\ t_1 & t_4 \\ \hline \end{array} \quad \gamma_1^4 = \begin{array}{|c|c|c|c|}\hline b & e & i & l \\ \hline b & e & i & l \\ t_2 & t_5 & t_9 & t_{13} \\ \hline \end{array} \quad \gamma_1^5 = \begin{array}{|c|}\hline \\ \hline \\ \\ \hline \end{array}$$

$$\gamma_1^6 = \begin{array}{|c|c|c|c|c|c|}\hline d & g & j & h & k & n \\ \hline d & g & j & h & k & n \\ t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} \\ \hline \end{array} \quad \gamma_1^7 = \begin{array}{|c|c|}\hline l & p \\ \hline l & p \\ t_{13} & t_{17} \\ \hline \end{array} \quad \gamma_1^8 = \begin{array}{|c|c|}\hline n & p \\ \hline n & p \\ t_{15} & t_{17} \\ \hline \end{array} \quad \gamma_1^9 = \begin{array}{|c|c|}\hline p & q \\ \hline p & q \\ t_{17} & t_{18} \\ \hline \end{array}$$

Figure 3.5. Subalignments between the trace $\sigma_1 = \langle a, b, e, i, l, d, g, j, h, k, n, p, q \rangle$ in event log L_1 and valid decomposition D_1 in Figure 3.3

All moves in alignments γ_3^1, γ_3^3, γ_3^6, γ_3^7, γ_3^8 and γ_3^9 are synchronized. Alignment γ_3^5 is an empty alignment. Alignments γ_3^2 and γ_3^4 both have a model move involving transition t_2 with the label b. Similarly, optimal subalignments for the traces σ_1 and σ_2 are shown in Figures 3.5 and 3.6 respectively.

A naive approach to aggregate the results per subcomponent, would be to sum up all the misalignment costs of the subalignments under the standard cost function. For

$$\gamma_2^1 = \begin{array}{|c|}\hline a \\ \hline a \\ t_1 \\ \hline \end{array} \quad \gamma_2^2 = \begin{array}{|c|c|}\hline a & c \\ \hline a & c \\ t_1 & t_3 \\ \hline \end{array} \quad \gamma_2^3 = \begin{array}{|c|c|}\hline a & d \\ \hline a & d \\ t_1 & t_4 \\ \hline \end{array} \quad \gamma_2^4 = \begin{array}{|c|}\hline \\ \hline \\ \\ \hline \end{array} \quad \gamma_2^5 = \begin{array}{|c|c|c|c|}\hline c & f & \gg & m \\ \hline c & f & \tau & m \\ t_3 & t_6 & t_{10} & t_{14} \\ \hline \end{array}$$

$$\gamma_2^6 = \begin{array}{|c|c|c|c|c|c|c|}\hline d & g & j & k & h & \gg & n \\ \hline d & g & j & \gg & h & k & n \\ t_4 & t_7 & t_{11} & & t_8 & t_{12} & t_{15} \\ \hline \end{array} \quad \gamma_2^7 = \begin{array}{|c|c|}\hline m & p \\ \hline m & p \\ t_{14} & t_{17} \\ \hline \end{array} \quad \gamma_2^8 = \begin{array}{|c|c|}\hline n & p \\ \hline n & p \\ t_{15} & t_{17} \\ \hline \end{array} \quad \gamma_2^9 = \begin{array}{|c|c|}\hline p & q \\ \hline p & q \\ t_{17} & t_{18} \\ \hline \end{array}$$

Figure 3.6. Subalignments between the trace $\sigma_2 = \langle a, c, f, m, d, g, j, k, h, n, p, q \rangle$ in event log L_1 and valid decomposition D_1 in Figure 3.3

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ \hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & c & f & \gg & m & d & g & j & k & h & \gg & n & p & q \\ \hline a & c & f & \tau & m & d & g & j & \gg & h & k & n & p & q \\ t_1 & t_3 & t_6 & t_{10} & t_{14} & t_4 & t_7 & t_{11} & & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & \gg & e & i & l & d & g & j & h & k & n & p & q \\ \hline a & b & e & i & l & d & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

Figure 3.7. Alignments for event log L_1

$\gamma_3^1, \gamma_3^2, \ldots, \gamma_3^9$, we would get a total of 2. However the misalignment costs associated to the optimal overall alignment γ_3 is 1 as shown in Figure 3.7. The wrong result is produced because border activities appear in multiple subnets and therefore moves involving these transitions will be counted multiple times when their associated costs are simply aggregated. We would like the result computed using $\gamma_3^1, \ldots, \gamma_3^9$ to equal the optimal cost computed using a overall alignment such as γ_3. Hence, we use the adapted cost function presented in [65], to avoid counting moves that involve border activities multiple times.

Definition 3.3 (Adapted cost function [65]). *Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of some system net S and $\delta \in A_M \to \mathbb{Q}$ a cost function. The adapted cost function $\delta_D \in A_M \to \mathbb{Q}$ for decomposition D is defined as follows:*

$$\delta_D(a,m) = \begin{cases} \frac{\delta(a,m)}{|S_b(\alpha(a,m),D)|} & \text{if } \alpha(a,m) \neq \tau; \\ \delta(a,m) & \text{otherwise.} \end{cases}$$

The cost of each legal move is divided by the number of subnets in which the corresponding activity may appear, for example, $\delta_D(a, \gg) = \frac{\delta(a,\gg)}{|S_b(a,D)|}$. This avoids counting misalignment costs of the same legal move multiple times. For example, consider the set of subalignments $\gamma_3^1, \ldots, \gamma_3^9$ in Figure 3.4, $|S_b(b, D_1)| = |\{S_1^2, S_1^4\}|$ and $|S_b(d, D_1)| = |\{S_1^3, S_1^6\}|$. For the adapted standard cost function δ_{D_1}, $\delta_{D_1}(\gg, (b, t_2)) = \frac{1}{2}$ and $\delta_{D_1}(d, (d, t_4)) = 0$. The aggregated cost of $\gamma_3^1, \ldots, \gamma_3^9$ is 1, i.e. identical to the costs of the overall optimal alignment γ_3 as shown in Figure 3.7.

Having defined the adapted cost function, the fitness values associated with the optimal subalignments per sublog and subnet can then be aggregated. This gives a decomposed fitness metric.

Definition 3.4 (Decomposed fitness metric). *Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$.*

Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S. For all $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i)$ is a subnet with an observable activity set $A_v^i = A_v(S^i)$.

For a log trace $\sigma_L \in L$, $\sigma_L^i = \pi_{A_v^i}(\sigma_L)$ is the projection of σ_L on the activity set of subnet S^i.

$$\mathit{fit}_D(\sigma_L, S, \delta) = 1 - \frac{\sum_{i \in \{1,\ldots,n\}} \delta_D(\lambda(\sigma_L^i, S^i))}{\mathit{move}_M(S) + \mathit{move}_L(\sigma_L)}$$

For an event log L, its decomposed fitness metric is computed as follows:

$$fit_D(L, S, \delta) = 1 - \frac{\sum_{\sigma_L \in L} \sum_{i \in \{1,\ldots,n\}} \delta_D(\lambda(\sigma_L^i, S^i))}{|L| \times move_M(S) + \sum_{\sigma_L \in L} move_L(\sigma_L)}$$

In the decomposed fitness metric, the misalignment costs of each subalignment are first aggregated using the adapted cost function. Afterwards, the total is normalized using the same value as the undecomposed relative fitness metric so that both metric values are normalized in the same manner. In the paper [65], it is shown that the decomposed fitness metric provides an upper bound to the fitness computed using the full alignment between the overall log and model.

Let us consider again the trace $\sigma_3 = \langle a, e, i, l, d, g, j, h, k, n, p, q \rangle$ and the valid decomposition D_1 in Figure 3.3. Assuming that for $S_1^1, \ldots, S_1^9$, λ gives the subalignments $\gamma_3^1, \ldots, \gamma_3^9$ as shown in Figure 3.4. The decomposed fitness metric between the trace and the subnets is computed as $fit_{D_1}(\sigma_3, S_1, \delta_1) = \frac{23}{24} \approx 0.958$. This is identical to the fitness value for the overall trace and the overall net.

Similarly, the formula can be applied to the log. Let the three subalignments shown in Figure 3.5, Figure 3.6 and Figure 3.4 be the optimal subalignments that correspond to the traces σ_1, σ_2 and σ_3. The decomposed fitness value of the event log L_1 and S_1 is $fit_{D_1}(L_1, S_1, \delta_1) = 1 - \frac{0\times20+2\times5+(\frac{1}{2}+\frac{1}{2})\times5}{12\times30+13\times20+12\times5+12\times5} = \frac{145}{148} \approx 0.980$. This is again identical to the fitness value for the overall log and the overall net. As such, the approach has decomposed the conformance checking problem whilst providing a conformance value for the overall log and net as output. However, this is not always the case, and cannot be generalized for the general case but only for cases satisfying specific properties. These properties relate to alignment moves corresponding to border activities.

3.4. Total border agreement and exact decomposed fitness

As mentioned earlier, the decomposed fitness does not always match the overall fitness metric in the general case. That is because the legal moves involving a particular activity may differ from one subnet to another. Let us consider the trace $\sigma_6 =$

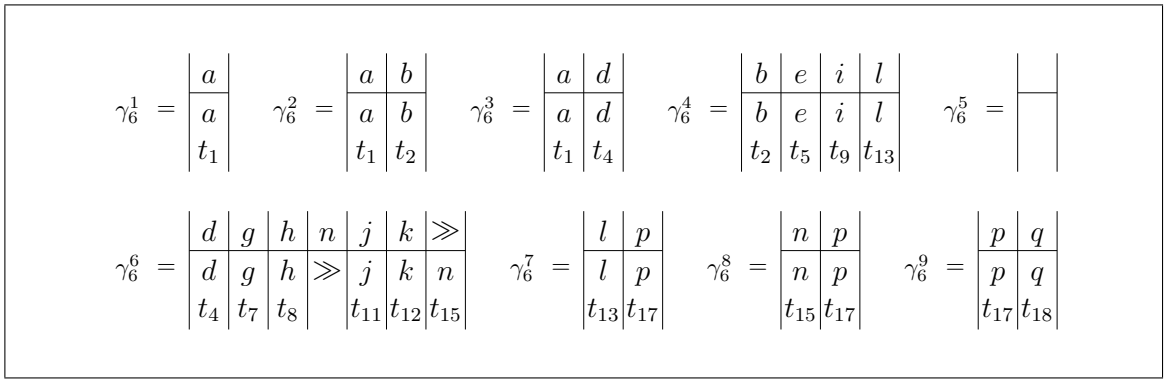


Figure 3.8. Subalignments between the trace $\sigma_6 = \langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$ in event log L_2 and valid decomposition D_1 in Figure 3.3

$\langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$ and the valid decomposition D_1 in Figure 3.3. A set of optimal subalignments between the trace and the subnets is shown in Figure 3.8. According to the system net S_1 in Figure 3.1, transition t_{15} with label n is to be executed after transitions t_{11} with label j and t_{12} with label k; in the trace σ_6, t_{15} is executed before t_{11} and t_{12}. This results in a log move of activity n at the fourth position and a model move of transition t_{15} with label n at the seventh position of alignment γ_6^6.

As activities j and k are not present in the subnet S_1^8, the move in the log and the move in the model are synchronized for transition n at alignment γ_6^8. Therefore, the moves involving border activity n are not identical between subalignments γ_6^6 and γ_6^8; the moves involving border activity n in the two subalignments are not in agreement. In this case, the decomposed fitness metric would not result in a value that is equal to the fitness value of the overall log and the overall net.

To compute the exact fitness value, a specific property must be satisfied: sequences of moves involving the same border activity have to be in agreement across all subalignments. We define that property as *border agreement* of subalignments and we formalize it as follows:

Definition 3.5 (Border agreement)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$.*

Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S. For all $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i)$ is a subnet with an observable activity set $A^i_v = A_v(S^i)$.

For a border activity $a \in A_b(D)$, let $a_M = \{(a,(a,t)), (\gg,(a,t)), (a, \gg)\}$ be the set of legal moves for activity a, where $t \in T$ such that $l(t) = a$.

Let $S^i \in S_b(a, D)$ be a subnet that has the border activity a. For a log trace $\sigma_L \in L$, $\sigma^i_L = \pi_{A^i_v}(\sigma_L)$ is the projection of σ_L on the activity set of S^i. $\gamma^i \in A^*_M$ denotes an optimal alignment between the sublog trace σ^i_L and some complete firing sequence of a subnet $\sigma^i_M \in \phi_f(S^i)$.

The set of subalignments $\gamma^1, \ldots, \gamma^n$ are under border agreement on a border activity $a \in A_b(D)$ if, and only if, $\pi_{a_M}(\gamma^i) = \pi_{a_M}(\gamma^j)$, for all $S^i, S^j \in S_b(a, D)$.

The set of subalignments $\gamma^1, \ldots, \gamma^n$ are under total border agreement (t.b.a.) if, and only if, border agreement is achieved one by one on all the border activities in $\gamma^1, \ldots, \gamma^n$ following the order of their occurrences across $\gamma^1, \ldots, \gamma^n$, starting with the first occurring border activity in subnet $S^i \in D$ where $I^i \geq I$.

Given the properties of a sequence, there is border agreement if the following three conditions are satisfied:

(i) $\pi_{a_M}(\gamma^i)$ has an equal number of moves as $\pi_{a_M}(\gamma^j)$.

(ii) $\pi_{a_M}(\gamma^i)$ has the same move types as $\pi_{a_M}(\gamma^j)$, i.e. if $\pi_{a_M}(\gamma^i)$ has one log move, then $\pi_{a_M}(\gamma^j)$ must also have one log move.

(iii) The order of moves in $\pi_{a_M}(\gamma^i)$ and $\pi_{a_M}(\gamma^j)$ are the same.

Note that if the subalignment γ^i is empty then the projection of the subalignment will also be an empty sequence.

For example, there is total border agreement between the valid decomposition D_1 in Figure 3.3 and the log trace $\sigma_3 = \langle a, e, i, l, d, g, j, h, k, n, p, q \rangle$. As shown by the corresponding subalignments in Figure 3.4, all the moves corresponding to each border activity are under border agreement.

Contrastingly, the subalignments of the trace $\sigma_6 = \langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$ are not under total border agreement. The moves involving border activity n in subalignment γ_6^6 is a log move followed by a model move which does not "agree" with the synchronous move in subalignment γ_6^8.

3.4.1. Properties of decomposed fitness

We now formalize the properties of the decomposed fitness metric with consideration to the border agreements of subalignments, i.e., if the total border agreement is not satisfied, the decomposed fitness metric corresponds only with an upper bound of the overall metric; when the total border agreement is satisfied, the decomposed fitness matches exactly the overall fitness. This results will be used in Sections 4.2 and 4.3 as part of the proposed conformance methods.

First, we note that the metric is an upper bound to the fitness metric computed using the full alignment between the overall log and model. This has been shown as Theorem 3 in the earlier paper [65].

However, under total border agreement, we can prove that the decomposed fitness value from the set of subalignments corresponds to the exact fitness value computed with the overall alignment. This applies to the decomposed log fitness value as well. We extend the properties of the decomposed fitness metric to include the capability of computing a conformance result that corresponds to an exact overall fitness value regardless of the conformance level.

Theorem 3.1 (Exact value for decomposed fitness metric under total border agreement)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log and let $\sigma_L \in L$ be a log trace. Let $S = (N, I, O) \in$*

$\mathcal{U}_S$ be a system net and let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S. For all $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i)$ is a subnet with an observable activity set $A_v^i = A_v(S^i)$. $\sigma_L^1, \ldots, \sigma_L^n$ are the subtraces from the projection of σ_L onto the activity sets of $S^1, \ldots, S^n$ such that $\sigma_L^i = \pi_{A_v^i}(\sigma_L)$. γ^i is an optimal subalignment between σ_L^i and some complete firing sequence of the corresponding subnet $\sigma_M^i \in \phi_f(S^i)$.

Let $\gamma^1, \ldots, \gamma^n$ be the set of subalignments and let them be under total border agreement. The decomposed fitness metric computed using this set of subalignments equals the relative fitness metric computed with the overall alignment between σ_L and S:

$$\mathit{fit}(\sigma_L, S, \delta) = \mathit{fit}_D(\sigma_L, S, \delta)$$

For the log L, if for all the log traces in L, their corresponding set of subalignments is under total border agreement,

$$\mathit{fit}(L, S, \delta) = \mathit{fit}_D(L, S, \delta)$$

PROOF. The paper [88] presents a stitching function that merges a set of subalignments into an optimal alignment if all the legal moves from the subalignments can be stitched together without conflict and a trace pseudo-alignment otherwise. We prove by contradiction that under total border agreement, the set $\gamma^1, \ldots, \gamma^n$ can always be stitched together as an optimal alignment without conflict.

Suppose that $\gamma^1, \ldots, \gamma^n$ is a set of subalignments and are under total border agreement but cannot be stitched together without conflict. The set of subalignments are comprised of only legal moves and therefore it must be that there is a stitching conflict for particular moves between the subalignments.

Let $(a_l, m) \in A_M$ be a move involved in the first conflict as $\gamma^1, \ldots, \gamma^n$ are being stitched together. As for a conflict we need another move, and hence another subnet,

we know that $|S_b(\alpha(a_l, m), D)| > 1$. Therefore, the activity $\alpha(a_l, m)$ must be a border activity. Under total border agreement, all subalignments for this activity are the same across all subalignments. Therefore, there cannot be a conflict.

Furthermore, since border agreement is achieved according to the occurrence order of the border activities across the subalignments, $\gamma^1, \ldots, \gamma^n$ can be stitched together without occurring any conflicts.

As a result, $\gamma^1, \ldots, \gamma^n$ can always be merged into an optimal alignment between σ and a complete firing sequence $\sigma_M \in \phi_f(S)$. With the merged optimal alignment γ, the sum of misalignment costs associated with $\gamma^1, \ldots, \gamma^n$ under the adapted cost function equals the misalignment cost of γ. Therefore, the decomposed fitness value with $\gamma^1, \ldots, \gamma^n$ equals the fitness value with γ.

The decomposed log fitness metric compares the sum of the misalignment costs from the sets of subalignments with the sum of the worst-case scenario costs for all the log traces. Since the set of subalignments corresponding to each log trace is under total border agreement, all sets of subalignments can be merged into optimal alignments. This means that the sum of the misalignment costs under the adapted cost function equals the sum of optimal misalignment costs associated with all the log traces. The decomposed log fitness value equals the log fitness value. □

As previously shown, the event log L_1 has the exact same value under both the decomposed fitness metric and the undecomposed fitness metric at about 0.980.

3.5. Limitations and extensions

While valid decompositions indeed yield sub-models and sub-logs whose activity sets is a subset of the overall activity set and that the overall model behavior is captured within the decomposed model behavior, the decomposed sub-models can actually allow more

behavior than before. This means that the A^* algorithm might have to explore a large number of irrelevant states to compute the optimal sub-alignment.

Consider sub-net S_1^6 from Figure 3.3. Within the context of the overall process model S, transition t_4 should be executed once in a complete model firing sequence unless the loop transition t_{16} is also executed during the firing sequence, in which case transition t_4 should be executed once for every firing of transition t_{16}. As such, projecting on activities d and o, the set of possible model traces should match with $d(od)*$ where we use the regular expression symbol $*$ to indicate that there can be zero or more od sub-sequences. However, if we consider the set of possible sub-traces permitted by sub-net S_1^6, transition t_{16} can always be fired because as a border activity, it does not have a input place to restrict its behavior.

3.5.1. Hide and reduce as an alternative replay approach

Instead of doing an edge partition to get a valid decomposition, in practice, the hide and reduce decomposed replay approach is used [84]. First, the hiding abstraction is applied to the net model given a subset of activities so that transitions that do not have an activity label in the given activity subset are made invisible. Figure 3.9 shows net S after the hiding abstraction given sub-net S_1^6's activity set $\{d, g, j, h, k, n\}$. After the hiding abstraction, one can elect to reduce the abstracted net using well-known Petri net reduction rules [51] so that redundant invisible transitions are reduced away.

Examining the hidden sub-net in Figure 3.9 shows that transition t_4 is no longer always enabled and instead can only be fired once per complete firing sequence from the initial marking $[p_1]$ to the final marking $[p_{19}]$ unless the loop transition t_{16} is fired as well. As such, in practice, we make use of this hide and reduce approach to perform decomposed alignment.

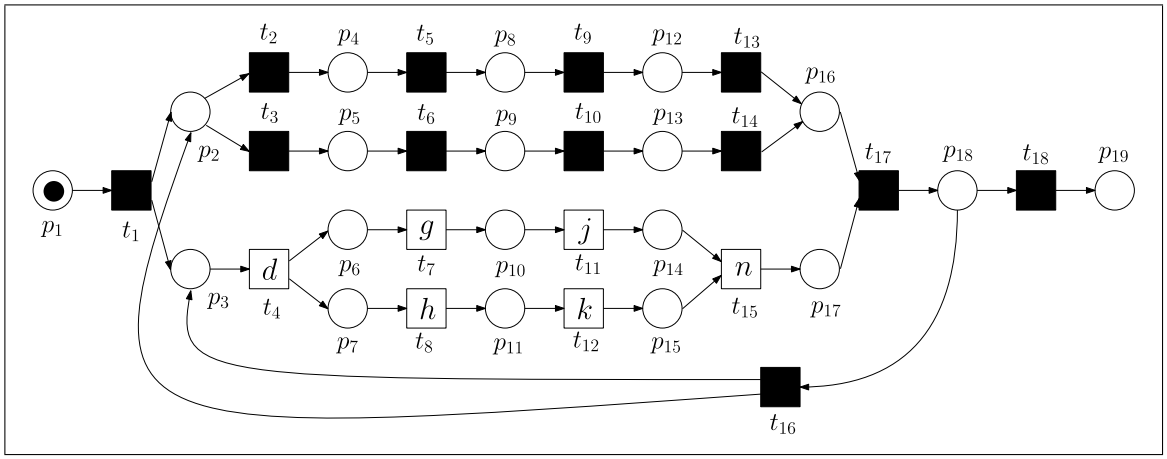


Figure 3.9. Hidden sub-net using activity subset of sub-net S_1^6 from Figure 3.3

3.6. Conclusion

This chapter presented the total border agreement condition as a sufficient condition so that merging sub-alignments gives an overall alignment that has the same misalignment costs as an optimal alignment computed under the monolithic approach. Furthermore, this chapter includes a discussion on how decomposed replay is done in practice to avoid creation of extra behavior by border activities that are always enabled. Next, we make use of the total border agreement property to design a conformance checking algorithm that computes alignments using the divide and conquer paradigm.

4. Recomposing conformance checking framework

4.1. Introduction

In the previous chapter, we presented the total border agreement as a sufficient condition for merging a set of decomposed sub-alignments to yield an overall alignment that has optimal misalignment costs. In this chapter we present an iterative approach that computes optimal alignments and thereby fitness in a divide and conquer manner. In a nutshell, this approach checks for the total border agreement condition for decomposed sub-alignments and resolves those which do not meet the condition using a coarser valid decomposition in the following iteration. This approach creates a full circle approach for decomposed alignment ('there and back again'), and makes it possible to solve alignment-based conformance problems that current techniques cannot handle. Importantly, our approach can balance quality and computation time. For example, in the experimental section, we demonstrate our approach on a real-life dataset (BPIC2012) for which the existing state-of-the-art monolithic conformance checking approach is not feasible while our proposed approach can compute an almost perfect approximation of the overall conformance in reasonable time, obtaining information about the specific problems of the model.

The methodology of this work can be seen as being under the paradigm of Design Science and it appropriately follows the seven accepted guidelines as presented in [28]. The resulting artifact of this work is a novel conformance checking approach that computes the overall fitness of a process and a log in a divide-and-conquer manner (Guideline 1). The motivating general problem relating to the rapid data growth and the limitations of the existing techniques have been aptly presented above (Guideline 2 and 6). The utility and efficacy of the artifact is demonstrated through mathematical proofs and empirical experiments (Guideline 3 and 5). The artifact and new insights drawn from it are produced as the contributions (Guideline 4). Lastly, we have taken care to present the ideas with a balanced level of detail so that the described artifact can be implemented (Guideline 7).

In this chapter, as the main contributions:

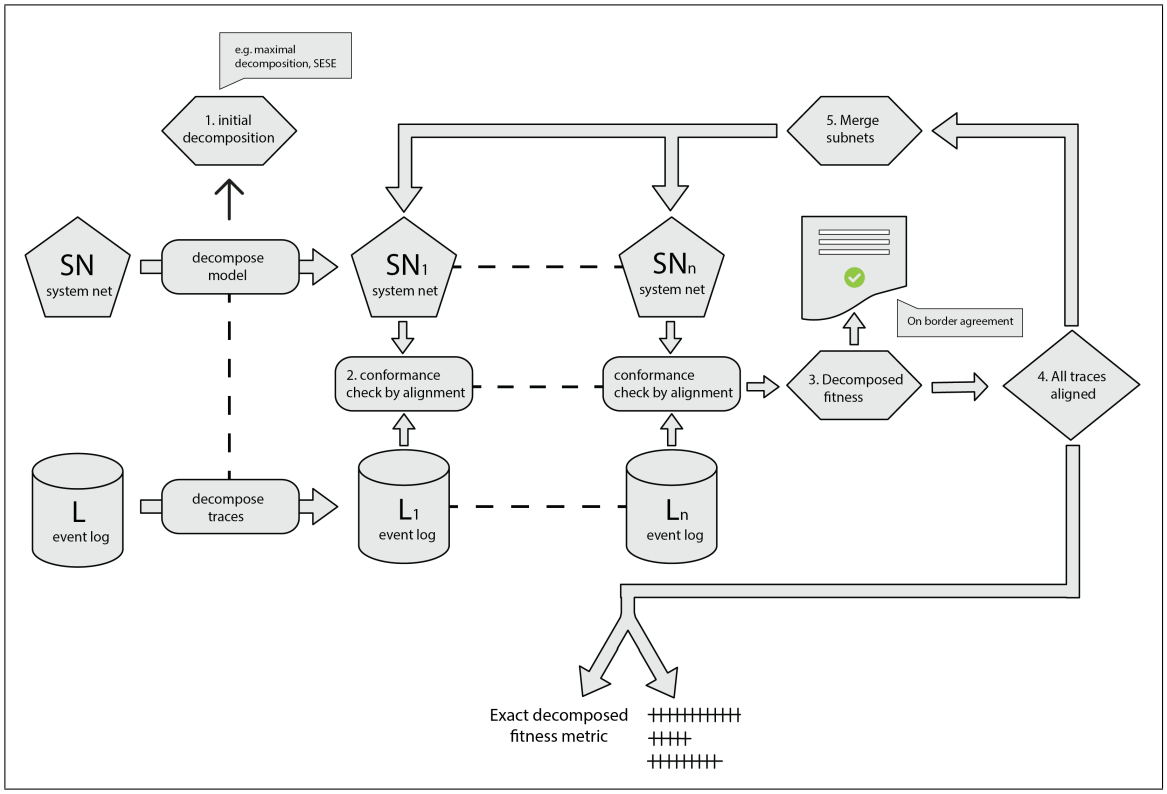


Figure 4.1. Overview of the exact decomposed conformance metric

- We present a novel decomposed conformance checking method – **recomposing conformance** – to compute the overall fitness between a process and a log (Section 4.2).
- Sometimes, exactness is not the top priority. We also modify the exact recomposing conformance method to compute an interval as an approximation of the overall fitness between a process and a log (Section 4.3).
- The two methods are implemented as one configurable conformance checking approach. The performance gains from the proposed methods are demonstrated through extensive experimental results using both synthetic and real-life datasets (Section 4.4).

4.2. Recomposing method for exact decomposed fitness

As previously mentioned, the main contribution of this work is to make decomposition techniques more applicable in computing the overall conformance between a net and log.

We now use the border properties formalized in the last section in a novel method to compute the overall conformance between a net and a log: *recomposing conformance*.

Similar to many existing decomposition techniques, the net is decomposed into subnets by activities and the log is projected onto the subnets to create sublogs. For each pair of subnet and sublog, alignments are created for the subnet and each subtrace in the sublog. Each of the sublogs can be analyzed in parallel, and together with the reduced size and complexity of the net, the approach obtains a significant performance gain in time [50, 91, 82, 87, 20]. Following the per subcomponent computation, the results are aggregated if there was agreement on the border. Otherwise, some disagreeing subnets are "recomposed" to get a more coarse-grained decomposition of the net, in which the disagreeing subnets have become a single subnet. As a result, they cannot disagree anymore. Traces whose conformance results disagreed on the border previously are recomputed under the new decomposition. This iterative process is repeated until completion. Figure 4.1 shows an overview of the summarized method. In this work, we apply this approach to decompose the relative fitness metric.

4.2.1. Decomposed fitness metric

As illustrated in Figure 4.1, the first step of the approach is to decompose the system net. This enables the performance gain in the conformance checking process. The decomposition of the system net has to fulfill the properties of a valid decomposition, first defined in [65].

Following the initial decomposition, the log is projected onto the subnets of the decomposition to get the sublogs for alignment.

The alignment of subnets and sublogs and the computation of the decomposed fitness metric are marked as step two and three of the algorithm in Figure 4.1. Decomposed fitness values of subalignments computed under total border agreement are recorded. Afterwards, their associated traces are taken out of the process and are marked as completed.

As for the remaining traces, we resolve their border agreement problems by selectively "recomposing" subnets by their matching border activities on which they disagree.

4.2.2. Subnet recomposition

As illustrated in step five in Figure 4.1, the existing set of subnets are recomposed as a new set of subnets. Recomposing subnets by their matching border activities resolves any border agreement problems associated with the recomposed border activities as it ceases to be shared between multiple subnets. "Recomposition" can be done on single or multiple border activities and different selection criteria can be used to select the border activities. For example, recomposing the subnets by multiple border activities is likely to resolve more border agreement problems than if the subnets were to be recomposed on only one border activity. However, there would be less performance gain under the multiple border activities selection approach as the resulting subnets would be larger and more complex.

t_1	t_2	t_3	t_4	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}
0	0	0	0	0	0	5	0	0

Figure 4.2. Vector showing the number of border agreement problems at each border activity for event log L_2

For the sake of simplicity, we consider selecting the single activity that has the highest number of border agreement problems. This selection criterion resolves the most problematic border activity at each recomposition. Following the valid decomposition in Figure 3.3, for event log L_2, there is a border agreement issue with trace $\sigma_6 = \langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$ on activity n. At the recomposition, activity n is identified and selected since it is the activity with the highest number of border agreement issues as shown in Figure 4.2. Retrieval of all the subnets that have the selected border activity is done by the shared function defined in Definition 3.2. These subnets are then merged, after which the set of recomposed subnets is a new valid decomposition of the system net.

Theorem 4.1 (Recomposition results in valid decomposition)**.** *Let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$. Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S.*

Let $a \in A_b(D)$ be a border activity that is shared between $|S_b(a, D)|$ subnets in D. Recomposing $S^1, S^2, \ldots, S^n$ on border activity a joins all the subnets in $S_b(a, D)$ on the activity a. Following the recomposition, which leads to a new decomposition D', $|S_b(a, D')| = 1$ and $a \notin A_b(D')$ i.e., a ceases to be a border activity.

Let $A' \subseteq A_b(D)$ be a subset of the border activities of D. Let D' be the recomposition of D on A', i.e., decomposition D' is recomposed on all activities $a \in A'$.

$D' \in \mathcal{U}_S$ i.e., D' is a valid decomposition of S.

PROOF. Recomposing $S^1, S^2, \ldots, S^n$ on a particular border activity $a \in A'$ joins the set of subnets $S_b(a, D)$ on activity a into one single subnet $S^+ = \bigcup S_b(a, D)$. S^+ has the same set of edges as $S_b(a, D)$ has. Therefore, $D' = (D \setminus S_b(a, D)) \cup \{S^+\}$ is a partitioning of the edges in S. Given that there was no creation of a new transition or a new place in the recomposition to S^+, it follows trivially that D' is a valid decomposition. The recomposition on any remaining border activities $a' \in A' \setminus \{a\}$ can be done in the same manner, and hence also yields a valid decomposition. □

4.2.3. New border agreement problems following recomposition

While recomposition can solve existing border agreement problems at merged border activities, new border agreement problems may arise at locations where there were no issues previously. Here we showcase such a case using the running example net S_1 in Figure 3.1 to explain the underlying intuition. This example is slightly more involved than the previous log examples in that it involves the loop construct of the net.

Consider trace $\sigma_7 = \langle a, b, e, i, d, g, j, h, k, b, e, i, l, d, g, j, h, k, n, p, q \rangle$. Replaying trace σ_7 on decomposition D_1 in Figure 3.3 would produce the set of subalignments as shown

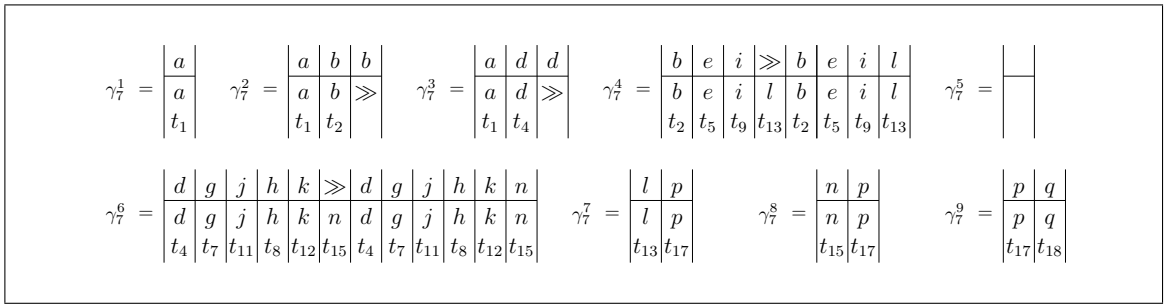


Figure 4.3. Subalignments between trace $\sigma_7 = \langle a, b, e, i, d, g, j, h, k, b, e, i, l, d, g, j, h, k, n, p, q \rangle$ and valid decomposition D_1 in Figure 3.3

in Figure 4.3. There are border agreement problems with border activities b, d, l, and n. This is caused by the fact that S_1^2, S_1^3, S_1^7, and S_1^8 are not aware that most of the subsequent activities of the branches associated to border activity b and d have been executed in trace σ_7. This indicates that border activities b, d, l, and n should not be marked as log moves. This shows that the information asymmetry or disparity between subnets on conformance is the cause of the border agreement problems. To resolve the border agreement problems, the subnets S_1^2, S_1^3. S_1^4. S_1^5. S_1^6. S_1^7. and S_1^8 are recomposed so that activity b, d, l, and n are no longer border activities. This produces the valid decomposition D_2 as shown in Figure 4.4. Replaying trace σ_7 on decomposition D_2 would produce the set of subalignments as shown in Figure 4.5. There are border agreement problems with border activities p. Similar to the previous iteration, this is caused by the fact that subnet S_1^{13} is not aware of the need for a model move on border activity p. One further recomposition on border activity p is required until the set of subalignments can be merged as the overall alignment as shown in Figure 4.6.

4.2.4. Iterative conformance checking

As such, at each iteration, a new valid decomposition is created from the recomposed subnets. Then, traces which had border agreement problems in the previous iteration are

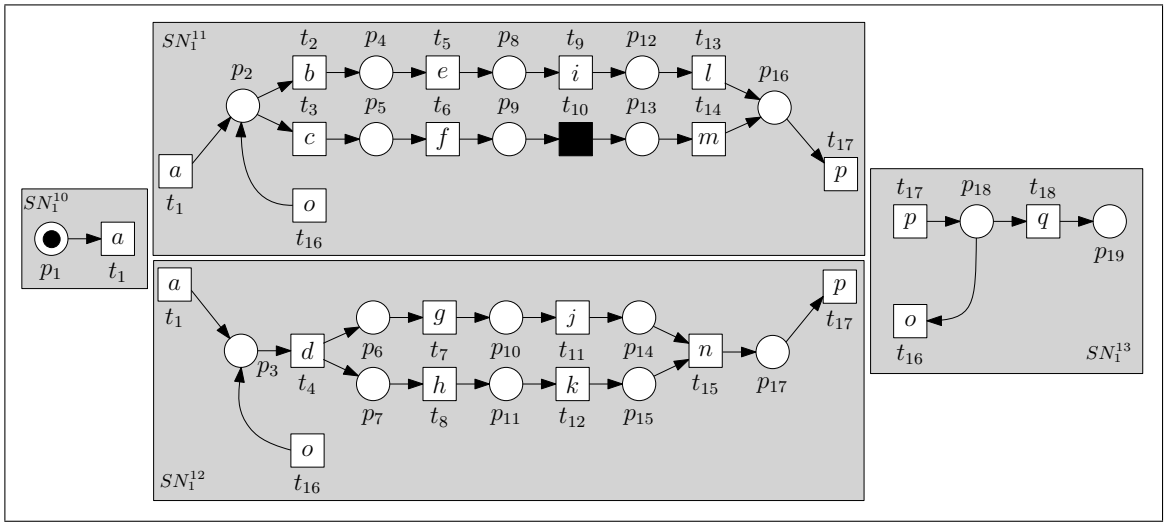


Figure 4.4. Valid decomposition D_2 of system net S_1 following the recomposition of subnets S_1^2, S_1^3. S_1^4. S_1^5. S_1^6. S_1^7 of decomposition D_1 in Figure 3.3

$$\gamma_7^{10} = \begin{array}{|c|}\hline a \\ \hline a \\ t_1 \\ \hline \end{array} \qquad \gamma_7^{12} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & d & g & j & h & k & \gg & \gg & \gg & d & g & j & h & k & n & p \\ \hline a & d & g & j & h & k & n & p & o & d & g & j & h & k & n & p \\ t_1 & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{16} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} \\ \hline \end{array}$$

$$\gamma_7^{11} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a & b & e & i & \gg & \gg & \gg & b & e & i & l & p \\ \hline a & b & e & i & l & p & o & b & e & i & l & p \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_{17} & t_{16} & t_2 & t_5 & t_9 & t_{13} & t_{17} \\ \hline \end{array} \qquad \gamma_7^{13} = \begin{array}{|c|c|}\hline p & q \\ \hline p & q \\ t_{17} & t_{18} \\ \hline \end{array}$$

Figure 4.5. Subalignments between trace $\sigma_7 = \langle a, b, e, i, d, g, j, h, k, b, e, i, l, d, g, j, h, k, n, p, q \rangle$ and valid decomposition D_2 in Figure 4.4

projected onto the new valid decomposition to be rechecked. Recomposition and conformance checking can be repeated until the decomposed fitness metric of all traces are computed under total border agreement.

$$\gamma_7 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & e & i & \gg & d & g & j & h & k & \gg & \gg & \gg & b & e & i & l & d & g & j & h & k & n & p & q \\ \hline a & b & e & i & l & d & g & j & h & k & n & p & o & b & e & i & l & d & g & j & h & k & n & p & q \\ t_1 & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{16} & t_2 & t_5 & t_9 & t_{13} & t_4 & t_7 & t_{11} & t_8 & t_{12} & t_{15} & t_{17} & t_{18} \\ \hline \end{array}$$

Figure 4.6. Alignment between trace $\sigma_7 = \langle a, b, e, i, d, g, j, h, k, b, e, i, l, d, g, j, h, k, n, p, q \rangle$ and system net S_1 in Figure 3.1

At completion, the set of alignments and the decomposed fitness value of the log are returned as output. This fitness value corresponds to the exact fitness value of the overall log and overall net.

While it can be crucial to have an exact fitness value of the overall log and net, in some scenarios an interval value may suffice. This is addressed in the next section.

4.3. Recomposing method for interval decomposed fitness

In some conformance checking scenarios, it may be sufficient to have an interval fitness value of the overall log and the overall net. For example, in the selection of candidates of genetic algorithms for the creation of a new generation of models, an interval value of each candidate model might already be sufficient to decide whether or not it should be kept. In addition, not requiring an exact fitness value can increase performance gain as there can potentially be less iterations of the recomposition and checking steps. Moreover, we note that while recomposition guarantees that any border agreement problems at the merged border activities will be gone in the next conformance checking iteration, new border agreement problems may arise at other border activities which had no problems previously. Lastly, it is possible that there are a few traces whose decomposed fitness value cannot be computed under total border agreement for many iterations or unless the overall trace and the overall net are used. As shown in Figure 4.7, the exact decomposed conformance algorithm is modified to compute an interval decomposed fitness value.

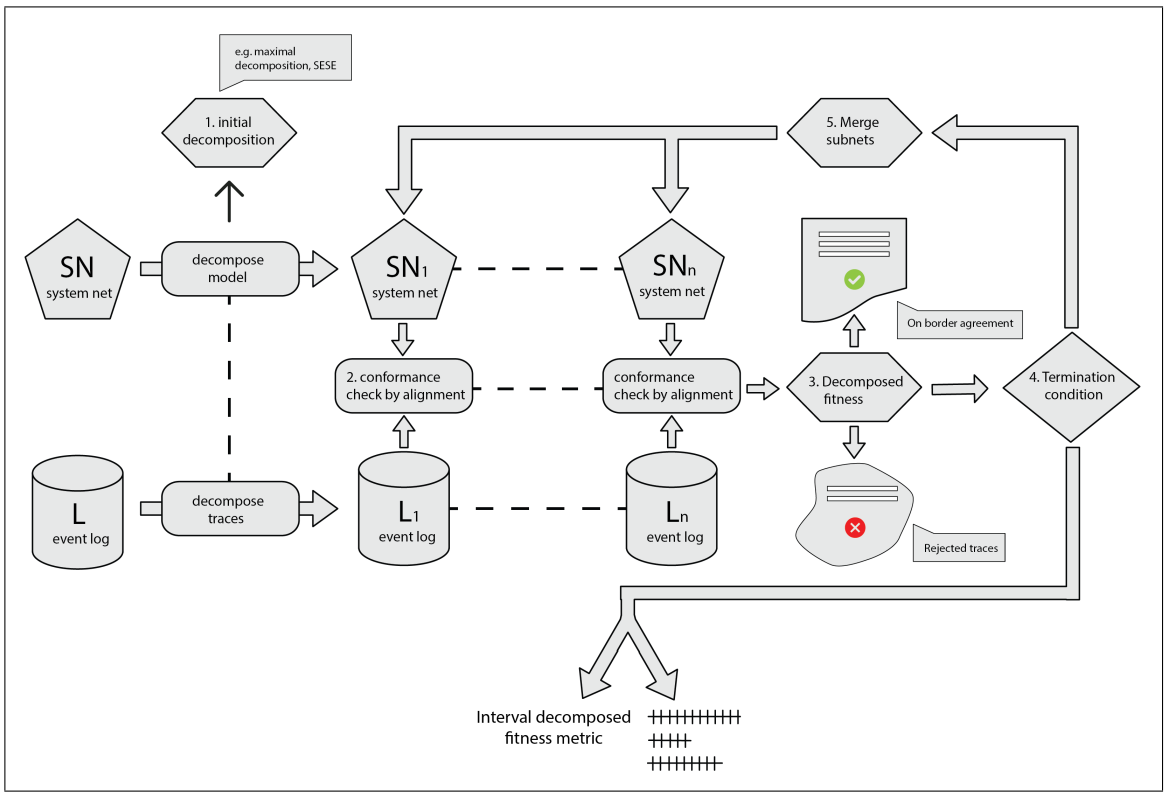


Figure 4.7. Overview of the interval decomposed conformance metric

4.3.1. Interval decomposed fitness conformance

As previously mentioned, the decomposed fitness metric has been shown to be an upper bound to the fitness metric of the overall log and the overall net. Using this property, an interval can be computed such that the fitness value of the overall log and the overall net is within the interval.

Definition 4.1 (Decomposed fitness interval)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$. Let $\sigma_L \in L$ be a log trace.*

Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S.

$$fit_D^{int}(\sigma_L, S, \delta) = [fit_D^{low}(\sigma_L, S, \delta), fit_D(\sigma_L, S, \delta)]$$

$fit_D^{low}(\sigma_L, S, \delta)$ defines the lower bound of the decomposed fitness interval such that it equals the decomposed fitness metric if the optimal subalignments $\gamma^1, \ldots, \gamma^n$ (where $\gamma^i = \lambda(\sigma_L^i, S^i)$) are under total border agreement (t.b.a.):

$$cost_D(\sigma_L, S, \delta) = \begin{cases} \sum_{i \in \{1,\ldots,n\}} \delta_D(\lambda(\sigma_L^i, S^i)) & \textit{if under t.b.a.}; \\ move_M(S) + move_L(\sigma_L) & \textit{otherwise}. \end{cases}$$

$$fit_D^{low}(\sigma_L, S, \delta) = 1 - \frac{cost_D(\sigma_L, S, \delta)}{move_M(S) + move_L(\sigma_L)}$$

The decomposed fitness interval of the event log L is computed as follows:

$$fit_D^{int}(L, S, \delta) = [fit_D^{low}(L, S, \delta), fit_D(L, S, \delta)],$$

$$fit_D^{low}(L, S, \delta) = 1 - \frac{\sum_{\sigma_L \in L} cost_D(\sigma_L, S, \delta)}{|L| \times move_M(S) + \sum_{\sigma_L \in L} move_L(\sigma_L)}$$

Consider the subalignments for the trace $\sigma_6 = \langle a, b, e, i, l, d, g, h, n, j, k, p, q \rangle$ in Figure 3.8. Due to the border agreement problem with activity n, the decomposed fitness interval for the trace is $fit_{D_1}^{int}(\sigma_6, S_1, \delta_1) \approx [0, 0.962]$. The decomposed fitness interval for event log L_2 is computed as $fit_{D_1}^{int}(L_2, S_1, \delta_1) \approx [0.815, 0.979]$. We note that for both the trace σ_6 and the log L_2, their fitness values are within the respective intervals: $fit(\sigma_6, S_1, \delta_1) = 0.923 \in [0, 0.962]$ and $fit(L_2, S_1, \delta_1) = 0.973 \in [0.815, 0.979]$.

Theorem 4.2 (Overall fitness is within interval fitness)**.** *Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$.*

Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S. For all $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i)$ is a subnet with an observable activity set $A_v^i = A_v(S^i)$.

For a log trace $\sigma_L \in L$, $\sigma_L^i = \pi_{A^i}(\sigma_L)$ is the projection of σ_L on the activity set of subnet S^i.

$$\mathit{fit}(\sigma_L, S, \delta) \in \mathit{fit}_D^{int}(\sigma_L, S, \delta)$$

For log L,

$$\mathit{fit}(L, S, \delta) \in \mathit{fit}_D^{int}(L, S, \delta)$$

PROOF. We prove the theorem by cases.

For $S^1, \ldots, S^n$, let $\gamma^1, \ldots, \gamma^n$ be the optimal subalignments where $\gamma^i = \lambda(\sigma_L^i, S^i)$ is the optimal subalignment of σ_L^i and S^i. There are two cases.

Case 1: $\gamma^1, \ldots, \gamma^n$ are under total border agreement

$\mathit{fit}_D^{int}(\sigma_L, S, \delta) = [\mathit{fit}_D(\sigma_L, S, \delta), \mathit{fit}_D(\sigma_L, S, \delta)]$. By Theorem 3.1, the fitness of the overall trace and net equals the decomposed fitness metric. Hence the overall fitness value is within the interval.

Case 2: $\gamma^1, \ldots, \gamma^n$ are not under total border agreement

$\mathit{fit}_D^{int}(\sigma_L, S, \delta) = [0, \mathit{fit}_D(\sigma_L, S, \delta)]$. The overall fitness is within the interval since the decomposed fitness metric computes an upper bound of the fitness value for the overall trace and net.

For the decomposed log fitness interval, the upper bound of the interval is the decomposed log fitness which has been shown to be an upper bound to the exact overall log fitness. The lower bound of the interval uses the total misalignment costs of the subalignments under the adapted cost function if there is total border agreement. Otherwise, it defaults to the worst-case scenario cost. In Theorem 3.1, it was proven that the exact misalignment cost is obtained if the set of subalignments is under total border agreement. This means that the sum of misalignment cost is an upper bound to the overall misalignment costs and therefore the lower bound of the decomposed log fitness interval is a lower

bound to the exact overall log fitness value. Hence, the exact overall log fitness is within the decomposed log fitness interval. □

As previously shown, the fitness value for the log L_2, $fit(L_2, S_1, \delta_1) \approx 0.973$ is within its decomposed fitness interval $fit^{int}_{D_1}(L_2, S_1, \delta_1) \approx [0.815, 0.979]$. To compute the interval decomposed metric, traces are exempted from the computation of an exact decomposed fitness value through trace reject and termination conditions.

4.3.2. Trace reject and termination conditions

As previously mentioned, it is possible that there are a few problematic traces which cannot be checked within a short time. As such, trace reject and termination conditions can be used to configure the balance between result quality and computation time.

Let $L \in \mathcal{B}(A^*)$ be an event log and let $S = (N, I, O) \in \mathcal{U}_S$ be a system net with $N = (P, T, F, l)$. Let $D = \{S^1, S^2, \ldots, S^n\} \in \mathcal{D}(S)$ be a valid decomposition of S. For all $1 \leq i \leq n$, $S^i = (N^i, I^i, O^i)$ is a subnet with an observable activity set $A^i_v = A_v(S^i)$.

For a log trace $\sigma_L \in L$, $\sigma^i_L = \pi_{A^i}(\sigma_L)$ is the projection of σ_L on the activity set of subnet S^i. For $1 \leq i \leq n$, γ^i is a subalignment between subnet S^i and subtrace σ^i_L. $\gamma^1, \ldots, \gamma^n$ is the set of subalignments corresponding to σ_L.

Following the computation of the decomposed fitness metric in step three of Figure 4.7, the traces in log L are partitioned over C, R, and B, i.e., $L = C + R + B$. Traces that are computed under total border agreement are added to the accepted traces multiset $C \in \mathcal{B}(A^*)$. Due to the number of border agreement issues or alignment time, traces that are not computed under total border agreement may be added to the rejected traces multiset $R \in \mathcal{B}(A^*)$. Otherwise, traces are added to the to-be-aligned multiset $B \in \mathcal{B}(A^*)$.Let the recomposition algorithm be at its kth iteration.

For trace σ_L, if its corresponding set of subalignments $\gamma^1, \ldots, \gamma^n$ is not under total border agreement, the trace is added to multiset R if,

- the number of border agreement issues of subalignments $\gamma^1, \ldots, \gamma^n$ is greater than the given threshold $x \in \mathbb{N}$, i.e., $|\{a \in A_b(D) \mid \exists_{1 \leq i < j \leq n} \pi_{\{a\}}(\gamma^i) \neq \pi_{\{a\}}(\gamma^j)\}| > x$.
- The time spent on aligning a subtrace σ_L^i with a subnet S^i is higher than the given threshold $y \in \mathbb{Q}$, e.g. a threshold of $y = 1$ millisecond per subalignment.

A rejected trace $\sigma'_L \in \{\sigma_L \in L \mid R(\sigma_L) > 0\}$ is not checked in future iterations. The criteria for the trace reject conditions can be adjusted to reflect the trade-off between result quality and computation time. A decomposed fitness interval is computed if $\exists_{\sigma_L \in L} R(\sigma_L) > 0$, i.e., at least one trace is rejected.

At the end of each iteration, termination conditions are examined to decide whether to proceed to the next iteration of the algorithm. If termination conditions are met, the decomposed fitness interval $fit_D^{int}(L, S, \delta)$ is returned. These termination conditions are defined on the log level. The termination conditions are as follows:

- All log traces have been either aligned under total border agreement or have been rejected, i.e., $C + R = L$.
- Surpassing the overall time threshold $z \in \mathbb{Q}$ for conformance checking, e.g., if more than $z = 1$ minute is needed in the conformance checking process.
- Having aligned a target percentage of traces $0 \leq v \leq 1$ in the log under total border agreement, i.e., $\frac{\sum_{\sigma_L \in L} C(\sigma_L)}{\sum_{\sigma_L \in L} L(\sigma_L)} \geq v$. For example, if more than $v = 0.9 = 90\%$ of the traces in the event log have been aligned under total border agreement.
- The overall fitness interval value $0 \leq w \leq 1$ is narrow enough, i.e., $fit_D(L, S, \delta) - fit_S^{low}(D, L, \delta) \leq w$. For example, if the interval range is less than $w = 0.1$.
- The maximum number of iterations $m \in \mathbb{N}$ is reached, i.e., $k \geq m$. For example, if $k = 100$ iterations have been done.

As illustrated in Figure 4.7, if the termination conditions are met in step four, the algorithm terminates and returns a decomposed fitness interval value $fit_D^{int}(L, S, \delta)$ and the alignments of traces whose decomposed fitness value had been computed under total border agreement.

4.4. Implementation and Evaluation

We have implemented the proposed conformance checking framework, and evaluated it on both artificial and real-life datasets. Our evaluations demonstrate the following main contributions:

(i) Recomposing conformance checking enables replays of model-log pairs that were previously not feasible under the monolithic approach. This increases the applicability of alignment-based conformance checking.

(ii) Logs associated to large models can take a tremendous amount of time to replay under the current monolithic approach. For these logs, the proposed recomposition strategy can lead to significant performance gains, speeding up the conformance checking process. The significant performance gains are present both in noiseless and noisy scenarios.

(iii) Recomposing conformance checking allows configurable approximations of conformance through interval fitness values. The computation time of interval fitness values is often shorter than the needed time for exact fitness values under both the monolithic and recomposition approach. This means that end users can get an idea of the conformance level within shorter times, or whenever there is a hard time constraint.

The section is structured as follows: First, we discuss the implementation details, the characteristics of the datasets used in the experiments, and the general conditions applied on the experimentation. Second, the exact fitness computation and the resulting speed-ups and improvement in the feasability are evaluated in detail, in both noiseless and noisy scenarios. Third, we illustrate the improvement in the feasibility of the alignment-based approach for different time-constrained scenarios using the recomposition strategy, and the effect of the time limits in the interval narrowing. Finally, the applicability of the approach for real-life scenarios is shown.

4.4.1. Implementation, datasets, and evaluations

The presented recomposing conformance checking framework has been implemented as the *Replay with Recomposition* plugin in the *DecomposedReplayer* package of ProM6.9[86]. *ProM* is an extensible framework that supports a wide variety of Process Mining techniques in the form of plug-ins. It is platform independent as it is implemented in Java, and can be downloaded free of charge.[1] Figure 4.9 shows a dialog box of the plugin at which the user can configure different values for the parameters introduced in the previous section, and Figure 4.8 shows the conformance analysis overview for one of the datasets tested. The implementation allows for the setting of all the parameters defined previously, and to manually select an initial decomposition for the approach (including the maximal decomposition if desired).

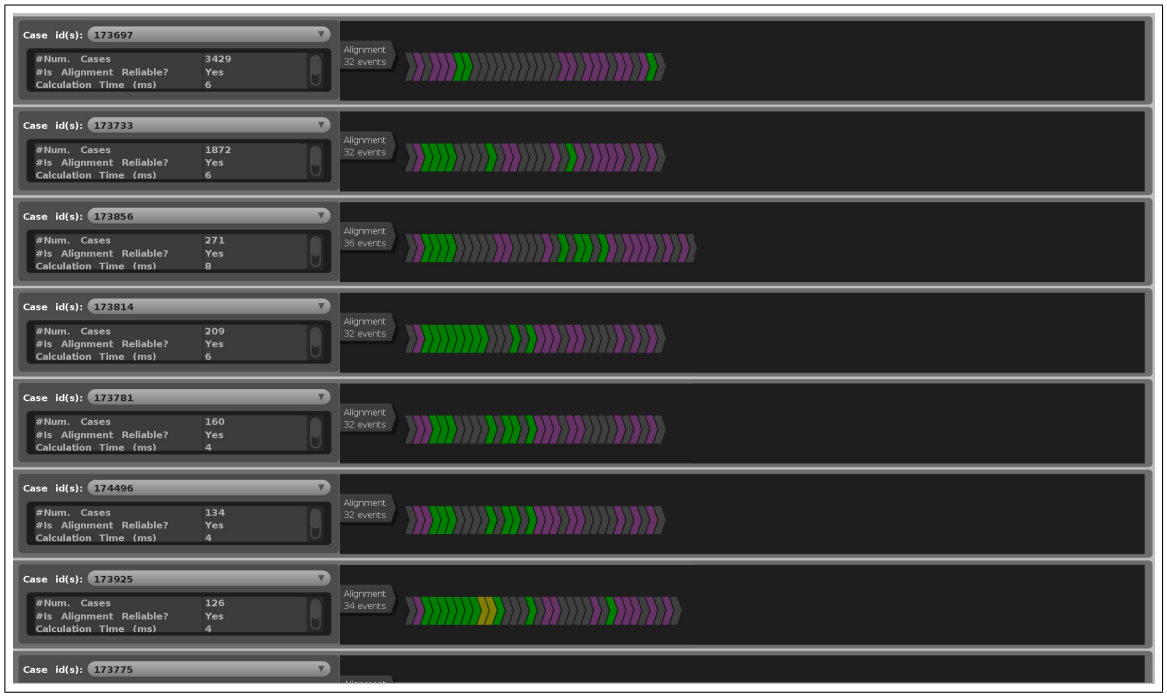


Figure 4.8. Resulting alignments for deviation diagnosis

[1] http://www.promtools.org/

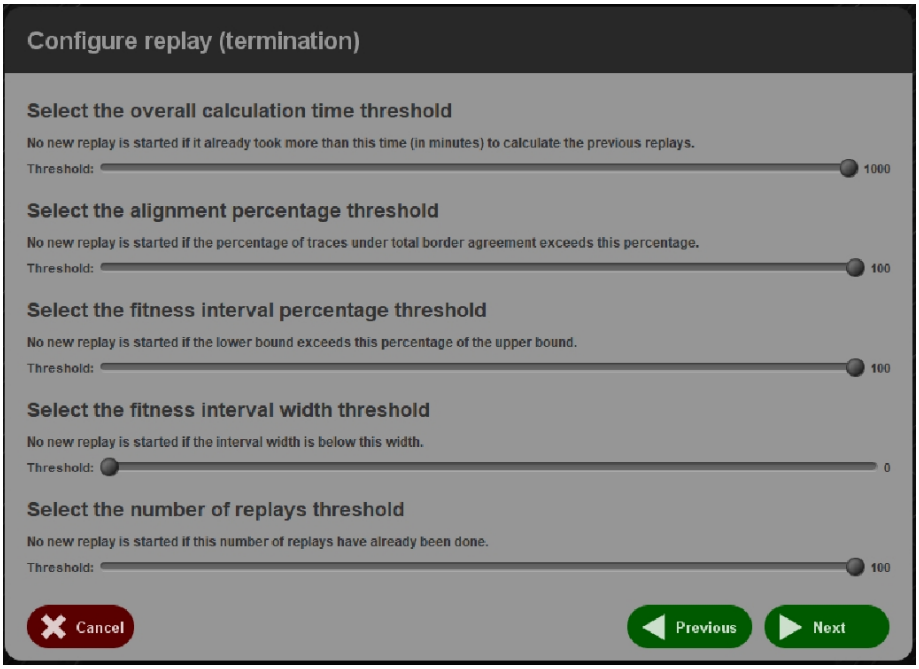


Figure 4.9. Dialog for *Replay using Recomposition*

Name	Source	$\|A\|$	|AND|	|XOR|	|Loop|	|Initial decomposition|
P241	synth	117	13	12	3	12
P246	synth	137	16	12	2	15
P272	synth	201	21	24	4	32
P275	synth	101	13	9	1	13
P284	synth	170	19	18	1	26
P285	synth	140	13	19	1	25
P291	synth	170	19	21	2	25
P297	synth	125	7	18	2	15
P307	synth	193	15	18	2	22
P313	synth	185	9	18	1	14
P347	synth	212	15	23	3	20
P381	synth	111	15	15	3	12
P383	synth	150	29	20	1	16
P430	synth	160	14	19	6	25
P436	synth	230	12	38	9	30

Table 4.1. Characteristics of the synthetic nets

The experiments were conducted including a wide-range of different datasets to represent the variability of possible scenarios. The datasets used include both synthetically generated datasets (represented as PX), and two real-life datasets (represented as BPIC201X).

The process models of the synthetic datasets were randomly generated using the PLG2 tool [14], and they include large processes containing combinations of all the most common workflow patterns, such as XOR, AND, loops, or invisible transitions. The model characteristics are shown in Table 4.1. Logs were generated from the models using simulation, and different operations were applied to emulate different plausible noise scenarios: no noise, noise by removing events and noise by swapping. This is explained in detail in Section 4.4.3. The names of the datasets refer to their time of creation and have no relation to any property of the processes such as the number of activities or arcs. Each log has 1000 cases. Following the Open Data Science principles, the datasets are open and publicly available [39]. The experimental evaluation also include two real-life datasets – based on the real cases BPIC2012 [72] and BPIC2017 [73] – to illustrate the applicability and benefits of the proposed approach for real cases. Details on the dataset are presented in Section 4.4.6, and the models and logs used are also publicly available.

The evaluations were performed on a desktop with two processors Intel E5-2470, 8 Cores, 2.3 Ghz, 32 GB RAM, running Linux 2.6.32 and using a 64-bit version of Java 7 where 6GB of RAM was allocated to the Java VM. Note that the approach can be distributed over a network of computers, but for the reported experiments, we only used one computing node. Although decomposed parts could be analyzed in parallel on a distributed system, this has not been implemented yet. Each experiment was conducted 3 times.

Parameters	MaxRecomposing
GlobalDurationThreshold	3600 seconds
LocalDurationThreshold	80 seconds
RelativeIntervalThreshold	100%
AbsoluteIntervalThreshold	0
MaxConflictThreshold	100
AlignmentPercentageThreshold	100%
MaxIterationThreshold	200

Table 4.2. MaxRecomposing configuration

In the evaluations, we experimented with different configurations adjusting the termination thresholds. The configuration with all the parameter values adjusted to the maximum will be identified as the *MaxRecomposing* configuration as shown in Table 4.2. For the sake of generality, this is the configuration shown in the results unless it is stated otherwise. Moreover, a *Hide and Reduce* projection strategy was used from the three net projection strategies proposed in [84]. Net projection is used to decompose the overall net into subnets.

As previously explained, the recomposition approach is instantiated with an initial decomposition. Different strategies can be used to decompose a model into subcomponents, e.g., maximal decomposition [65], SESE-based decomposition [50], passage-based decomposition [64], or cluster-based decomposition [89]. For the sake of simplicity, the experiments use a manual decomposition where the decompositions are manually decided through visual inspection of the model. As shown in Figure 4.10, the partitions follow closely to a SESE-based decomposition [50]. As such, the initial decomposition is done without previous knowledge of the locations where there are conformance issues. This is the default decomposition method unless stated otherwise. The number of subnets at the initial decomposition of each model is shown in Table 4.1.

4.4.2. Exact fitness in noiseless scenarios

We first experimented with the simplest scenario where the model and log are perfectly fitting. In this section we present conformance checking results for logs without noise ("no-noise" logs). The goals of this section are:

- To compare the feasibility of the recomposition approach with the existing monolithic approach.
- To compare and analyze computation times of replays under both approaches.

The synthetic process models and "no-noise" logs generated by the PLG2 tool are used in the experiments. "No-noise" logs are perfectly fitting with the corresponding models.

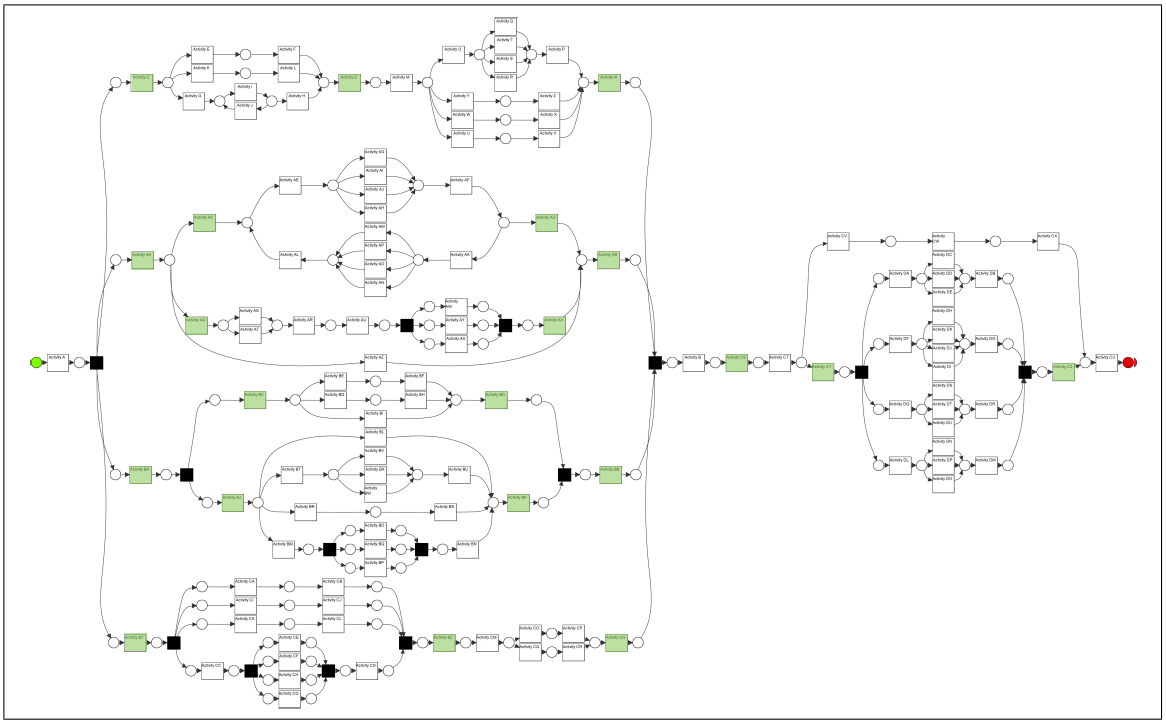


Figure 4.10. Initial manual decomposition of model P297 where border transitions are colored in green

This means that all the behaviour observed in the log can be matched to the behaviour modeled by the model so that the log can be replayed perfectly on the model and the fitness value equals to 1. We note that this is the least computationally intensive of all possible scenarios since the alignment algorithm can always match a log trace with a corresponding model trace as an alignment of solely synchronous moves. This means that the iterative process of the recomposition approach does not occur for the replays of these model-log pairs because moves associated with border transitions are always synchronous moves and in agreement.

There are 15 model-log pairs and for each dataset we conduct two experiments using the recomposition approach and the monolithic approach respectively. For each experiment, a 1 hour time limit is set such that replays which are still running after the time limit are stopped and deemed infeasible. Other than the replay feasibility, fitness, time, average speedup, and median speedup, we also report the number of activities ($|A|$) in the models,

Dataset							Monolithic			Recomposition				
Name	Source	Noise	$\|A\|$	$\overline{\|\sigma\|}$	$\|L\|$	$\|\{\sigma \in L\}\|$	Feasible	Fitness	$\overline{\text{Time (s)}}$	Feasible	Fitness	$\overline{\text{Time (s)}}$	$\overline{\text{Speedup}}$	$\widetilde{\text{Speedup}}$
P241-L36	synth	no	117	95	1000	1000	✓	1	398	✓	1	21	18.6	14.6
P246-L9	synth	no	137	77	1000	1000	✓	1	250	✓	1	23	11.1	8.8
P272-L30	synth	no	201	101	1000	1000	✓	1	645	✓	1	26	24.4	19.3
P275-L0	synth	no	101	77	1000	967	✓	1	167	✓	1	23	7.2	6.0
P284-L33	synth	no	170	102	1000	570	✓	1	891	✓	1	30	29.6	24.3
P285-L12	synth	no	140	46	1000	967	✓	1	103	✓	1	22	4.6	5.1
P291-L21	synth	no	170	90	1000	1000	✓	1	904	✓	1	26	35.1	41.5
P297-L6	synth	no	125	59	1000	1000	✓	1	236	✓	1	22	10.6	12.3
P307-L27	synth	no	193	22	1000	572	✓	1	57	✓	1	23	2.5	2.7
P313-L24	synth	no	185	20	1000	480	✓	1	68	✓	1	23	3.0	3.4
P347-L39	synth	no	212	60	1000	1000	✓	1	280	✓	1	26	10.6	12.1
P381-L3	synth	no	111	73	1000	978	✓	1	95	✓	1	26	3.7	5.2
P383-L15	synth	no	150	108	1000	631	✓	1	193	✓	1	23	8.3	9.3
P430-L18	synth	no	160	104	1000	1000	✓	1	291	✓	1	24	12.1	13.7
P436-L45	synth	no	230	35	1000	621	✓	1	109	✓	1	24	4.5	4.9

Table 4.3. Replay feasibility and computation times for synthetic logs without noise

the number of log traces ($|L|$), the number of trace variants ($|\{\sigma \in L\}|$), and the average trace lengths ($\overline{\sigma}$) of the logs.

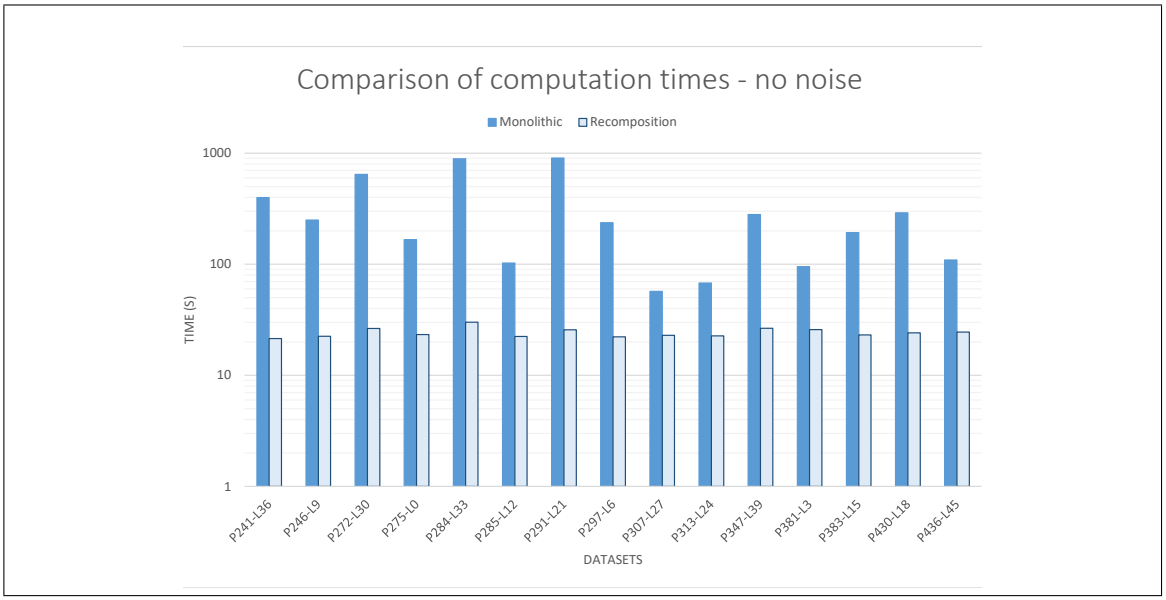


Figure 4.11. Feasible computation times for synthetic logs without noise

The results are presented in Table 4.3. We first note that the results from feasible replays are, as expected, all equal to 1, i.e., both approaches reported the conformance of the datasets correctly. Replay is feasible for all datasets under both approaches. Figure 4.11

compares the computation times of the monolithic replays with the computation times of the recomposing replays on a **logarithmic scale**. The figure shows that there is a clear performance gain in time across all datasets under the recomposition approach. Furthermore, the total computation times under the recomposition approach have much lower variability than the monolithic approach. Computation times range from 21s to 30s under the recomposition approach while they range from 57s to 904s under the monolithic approach. The stability in computation time under the recomposition approach allows its users to be more certain about the required execution time.

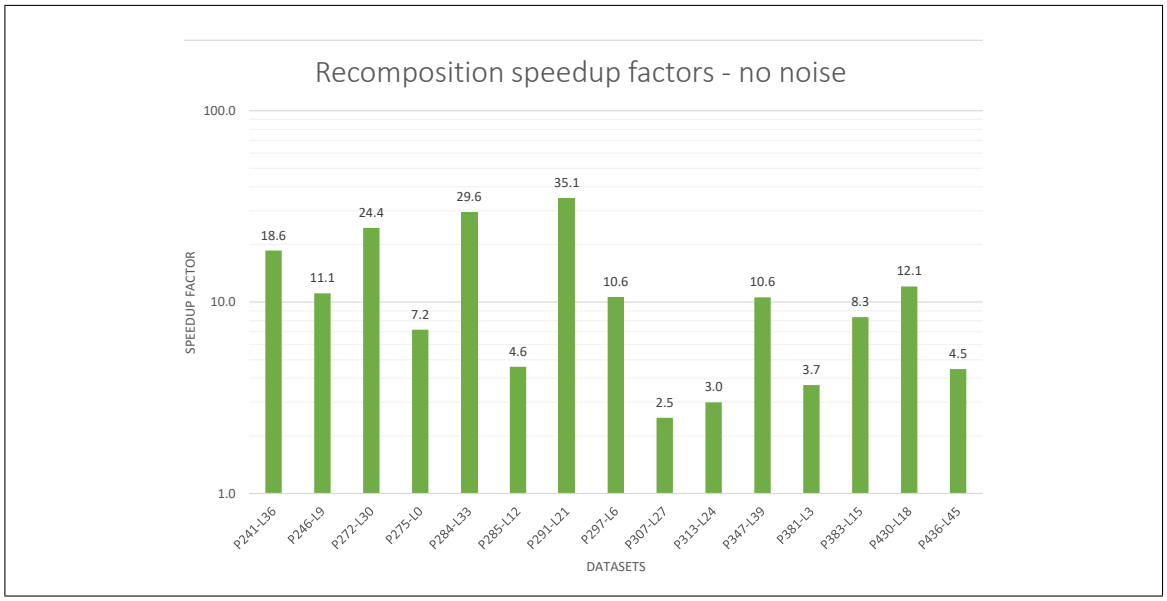


Figure 4.12. Speedup factors from recomposition approach over monolithic approach for synthetic logs without noise

The speedup factor per dataset is presented in Figure 4.12 where we see that it ranges from 2.5× to 35.1×. A 2.0× speedup factor means that the recomposition approach is twice as fast as the monolithic approach and a 10.0× speedup factor means that the recomposition approach is ten times faster.

Furthermore, speedup factors with respect to the average trace length of the log are shown in Figure 4.13. We can observe that datasets with longer average trace lengths generally have greater speedup factors. While there are differences between the average

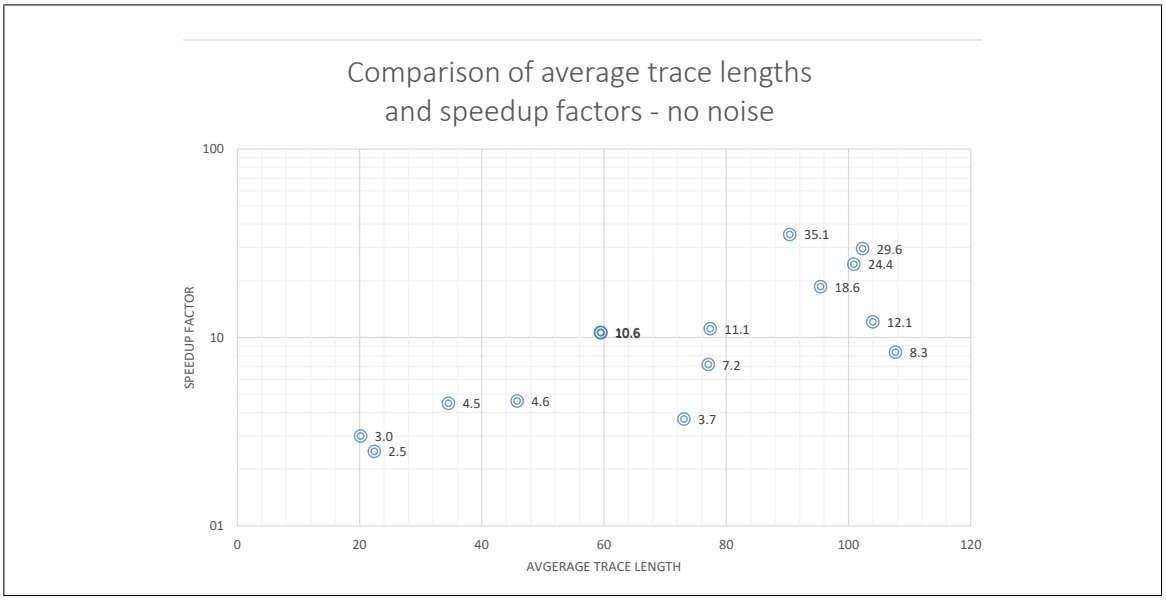


Figure 4.13. Speedup from recomposition approach in relation to average trace length for synthetic logs without noise

speedups and the median speedups, they do not significantly affect the presented conclusions.

In terms of feasibility, both the monolithic and recomposition approach have the same performance as they were able to complete all replays. However, the results show a significant improvement in computation time using the recomposition approach. As shown in [50], the speedup results from the decomposition of the alignment problem itself and the trace groupings formed by the decomposition. Smaller components usually take significantly less time to replay than large components such that replaying several subcomponents of a decomposition is faster than replaying the original component. Furthermore, distinct traces can share identical subsections. Trace groupings from decomposition can be these subsections such that the previously distinct traces become trace groupings where some are identical. Since alignments are not recomputed for the same trace, the number of alignment operations can be reduced.

In real-life cases, it is likely that the observed and modeled behaviour are not perfectly fitting. Therefore, further experiments are conducted using logs with noise, i.e., the log is not perfectly fitting with the model.

4.4.3. Exact fitness in noisy scenarios

In this section, we present conformance checking results for logs with noise. This means that there are discrepancies between the modeled and observed behaviour such that the fitness value is less than 1. Similar to the previous section, the goals of this section are:

- To compare the feasibility of the proposed recomposition approach with the existing monolithic approach.
- To compare and analyze computation times of replays under both approaches.

For the sake of comparison, the processes and models are the same as the previous section, but noise is included in the logs at their generation. To test the performance of the two approaches in detecting different types of deviations, we experimented with two noise types by including them into two separate logs. This means that each model is associated with two "noisy" logs. Each noise type mimics a specific scenario where there is a mismatch between modeled and observed behaviour. Alignments between log traces and net are computed on synchronous product nets which combine the two; different noise characteristics in the log produce different log traces which can have different impacts on the performance of alignment computation. The characteristics of the noise types and the process of generating these "noisy" logs can be summarized as follows:

- *MissingTrace* noise is where a trace has a probability of missing a part of its head, tail and/or episode. The noise is included into the log by log generation using the PLG2 tool. The specific configuration used for the log generation is shown in Table 4.4 (we refer to [14] for a detailed explanation of each parameter).
- *Swapped* noise is where two events in a trace can be inverted. This noise type mimics the scenario where the process has a specific location where deviation always occurs.

We generate a log with *Swapped* noise by swapping two activities at a specific location in the model before generating a noiseless log using the modified model. The point of deviation is randomly chosen but it is ensured that cases always pass through this location.

Parameters	MissingTrace
Number of traces	1000
Integer data object error probability	0%
Integer data object error delta	0
String data object error probability	0%
Change activity name probability	0%
Trace missing head probability	0.1%
Head max size	2
Trace missing tail probability	0.1%
Tail max size	2
Trace missing episode probability	0.1%
Episode max size	2
Perturbed event order probability	0%
Doubled event probability	0%
Alien event probability	0%

Table 4.4. PLG2 log generation configurations for *MissingTrace* dataset

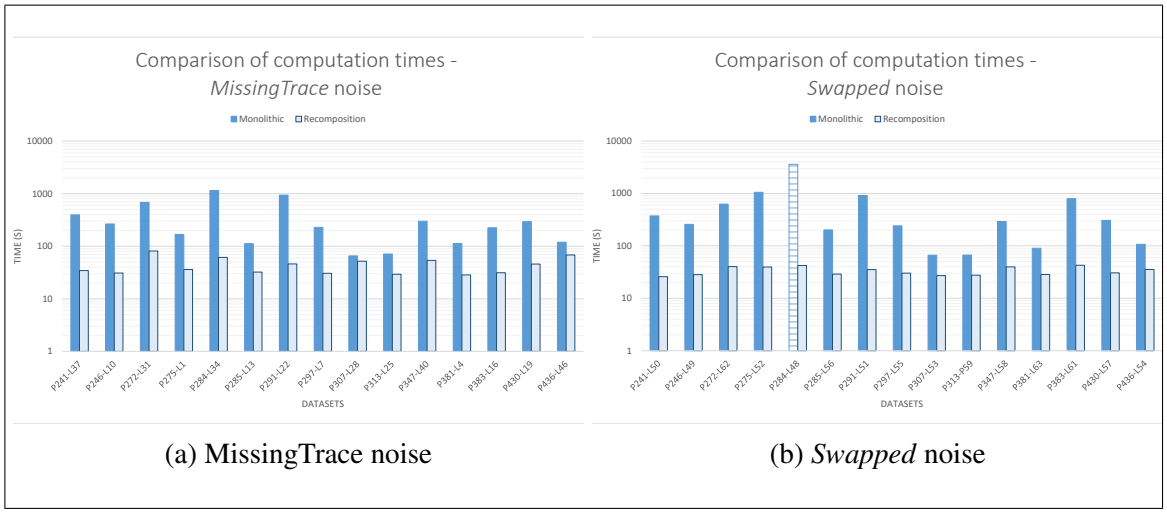
There are 30 model-log pairs and for each dataset we conduct two experiments using the recomposition approach and the monolithic approach. At each experiment, a 1 hour time limit is set such that replays which are still running after the time limit are stopped and deemed infeasible. Other than the replay feasibility, fitness, time, average speedup, median speedup, and the number of recompositions, we also report the number of transitions in the models, the number of unique log traces, and the average trace lengths of the logs.

Table 4.5 presents the results. On the whole, monolithic replay was not feasible for 1 model-log pair (P284-L48) due to the 1 hour time limit. At the time column, the infeasible replay is marked with "> 3600".

Figure 4.14 compares the computation times of monolithic replays and recomposing replays for logs with *MissingTrace* noise and *Swapped* noise. Computation times are

Dataset							Monolithic			Recomposition					
Name	Source	Noise	$\|A\|$	$\overline{\|\sigma\|}$	$\|L\|$	$\|\{\sigma \in L\}\|$	Feasible	Fitness	$\overline{\text{Time (s)}}$	Feasible	Fitness	$\overline{\text{Time (s)}}$	$\overline{\text{Recompose}}$	$\overline{\text{Speedup}}$	$\widetilde{\text{Speedup}}$
P241-L37	synth	miss	117	94	1000	1000	✓	0.999	395	✓	0.999	34	5	11.5	9.8
P241-L50	synth	swap	117	95	1000	1000	✓	0.989	374	✓	0.989	26	1	14.4	11.7
P246-L10	synth	miss	137	77	1000	1000	✓	0.998	265	✓	0.998	31	2	8.6	6.9
P246-L49	synth	swap	137	77	1000	1000	✓	0.987	257	✓	0.987	28	1	9.1	7.6
P272-L31	synth	miss	201	101	1000	1000	✓	0.999	685	✓	0.999	81	15	8.4	8.5
P272-L62	synth	swap	201	101	1000	1000	✓	0.989	626	✓	0.989	40	1	15.6	13.8
P275-L1	synth	miss	101	73	1000	984	✓	0.998	168	✓	0.998	36	9	4.6	3.9
P275-L52	synth	swap	101	74	1000	965	✓	0.984	1052	✓	0.984	39	1	26.6	21.6
P284-L34	synth	miss	170	106	1000	708	✓	0.998	1147	✓	0.998	61	10	18.7	23.1
P284-L48	synth	swap	170	106	1000	589	✓		> 3600	✓	0.997	42	1	> 85.5	> 85.9
P285-L13	synth	miss	140	46	1000	978	✓	0.997	112	✓	0.997	32	6	3.5	3.9
P285-L56	synth	swap	140	48	1000	973	✓	0.947	202	✓	0.947	29	0	7.0	8.0
P291-L22	synth	miss	170	90	1000	1000	✓	0.999	935	✓	0.999	46	6	20.3	21.5
P291-L51	synth	swap	170	91	1000	1000	✓	0.989	913	✓	0.989	35	1	25.9	30.1
P297-L7	synth	miss	125	59	1000	1000	✓	0.998	228	✓	0.998	31	4	7.5	7.7
P297-L55	synth	swap	125	60	1000	1000	✓	0.982	241	✓	0.982	30	1	8.0	9.2
P307-L28	synth	miss	193	22	1000	660	✓	0.994	65	✓	0.994	52	11	1.3	1.4
P307-L53	synth	swap	193	23	1000	581	✓	0.996	66	✓	0.996	27	0	2.4	2.3
P313-L25	synth	miss	185	20	1000	534	✓	0.993	71	✓	0.993	29	1	2.4	2.6
P313-L59	synth	swap	185	20	1000	465	✓	0.986	67	✓	0.986	28	0	2.4	2.5
P347-L40	synth	miss	212	60	1000	1000	✓	0.998	298	✓	0.998	54	9	5.5	6.2
P347-L58	synth	swap	212	59	1000	1000	✓	0.996	292	✓	0.996	40	1	7.3	8.5
P381-L4	synth	miss	111	73	1000	983	✓	0.998	112	✓	0.998	28	1	3.9	4.6
P381-L63	synth	swap	111	74	1000	973	✓	0.985	90	✓	0.985	28	1	3.2	3.4
P383-L16	synth	miss	150	106	1000	737	✓	0.999	224	✓	0.999	31	2	7.1	8.4
P383-L61	synth	swap	150	111	1000	632	✓	0.979	795	✓	0.979	43	1	18.7	19.8
P430-L19	synth	miss	160	104	1000	1000	✓	0.999	293	✓	0.999	46	10	6.4	7.2
P430-L57	synth	swap	160	104	1000	1000	✓	0.997	306	✓	0.997	30	0	10.0	11.2
P436-L46	synth	miss	230	35	1000	735	✓	0.996	120	✓	0.996	68	12	1.8	2.0
P436-L54	synth	swap	230	35	1000	633	✓	0.982	107	✓	0.982	35	1	3.0	3.3

Table 4.5. Replay feasibility and computation times for synthetic logs with *MissingTrace* and *Swapped* noise

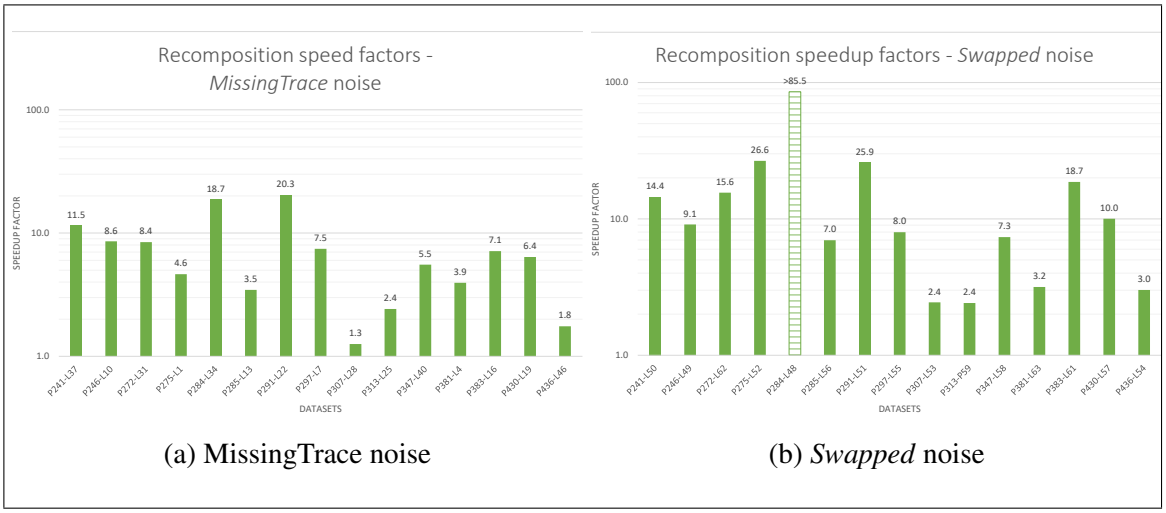


(a) MissingTrace noise

(b) *Swapped* noise

Figure 4.14. Feasible computation times for synthetic logs (infeasible replays are shown using a dashed pattern instead of a solid fill)

shown on a **logarithmic scale** in both figures. There are clear performance gains from adopting the recomposition approach.



(a) MissingTrace noise

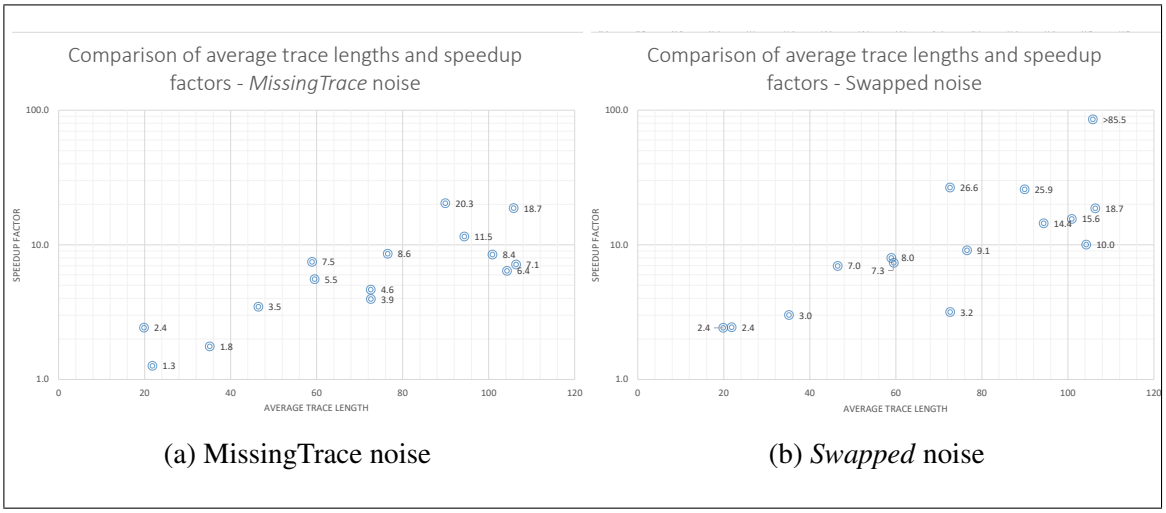
(b) *Swapped* noise

Figure 4.15. Speedup factors from recomposition approach over monolithic approach for synthetic logs (infeasible replays are shown using a dashed pattern instead of a solid fill)

Figure 4.15 shows the speedup factors for logs with *MissingTrace* noise and the speedups for logs with *Swapped* noise. As previously explained, a 2.0× speedup means that the recomposition approach is twice as fast as the monolithic approach and a 10.0× speedup means that it is 10× faster. We observe that speedups from the recomposition approach range from 1.3× to at least 85.5×. For the particular model-log pair P275-L52, computation time was reduced by a factor 26.6 from 17.5 minutes (1052 seconds) under the monolithic approach to 39 seconds under the recomposition approach.

Speedup factors in relation to the average trace length of logs with *MissingTrace* noise and *Swapped* noise are shown in Figure 4.16. We can observe that in general logs with longer average trace lengths have greater speedup factors.

In terms of feasibility, the recomposition approach outperforms the monolithic approach in replaying logs with *MissingTrace* or *Swapped* noise as the monolithic approach was infeasible for 1 model-log pair while the recomposition approach was feasible for all datasets. More importantly, for these model-log pairs, the recomposition approach was able to compute their exact fitness values in very little time (less than 2 minutes).



(a) MissingTrace noise

(b) *Swapped* noise

Figure 4.16. Speedup from recomposition approach in relation to average trace length for synthethic logs

The positive relationship between average trace lengths and speedups reflects the advantage of the recomposition approach. Logs with long average trace lengths tend to be associated with models that are more complex and therefore more challenging when computing alignments. As previously explained, the decomposition of a complex component into several decomposed subcomponents reduces replay time significantly. However, current techniques only guarantee that conformance results from decomposed subcomponents can be aggregated to an overall result if the model and log are perfectly fitting. By formalizing border activities and border agreement, we relax the strict requirement for perfect fitness and allow the aggregation of results from decomposed subcomponents even when there are mismatches between the model and log. This means that the recomposition approach gets the performance gains from decomposition and the resulting exact fitness result would correspond to the fitness value that one would get using the monolithic approach. The increasing speedups as average trace length increases demonstrate the said performance gains from decomposition despite the presence of noise in the log. This is further supported by observing the minimal effect that the average trace length has on the computation time of the recomposing replays. Analyzing the number of recompositions,

we note that the larger and more complex datasets require more recompositions. For example, P272-L31 required 15 recompositions while P241-L37 required 5 recompositions. Also, the noise type has a significant influence on the number of recompositions regardless of the model characteristics. Experiments on dataset with missing noise never require more than one recomposition while experiments on dataset with missing noise can require a maximum of 15 recompositions. These two points are further examined and explained in Section 4.4.4. While there are differences between the average speedups and the median speedups, they do not significantly affect the presented conclusions.

In conclusion, these experiments empirically show the potential performance gains of the recomposition approach. As the results illustrate, this new approach not only increases replay feasibility, but also results in significant performance gains for datasets that are feasible under both recomposition and monolithic approaches.

4.4.4. Bottlenecks for the monolithic and recomposition approach

One principal motivation of investigating decomposition techniques for alignment-based conformance checking is to improve the alignment computation time for large and complex processes. It has been previously shown that by decomposing the conformance problem into parts, this bottleneck can be alleviated significantly [50]. This section evaluates the bottlenecks of both approaches by analyzing the computation times of all the previous experiments. The computation time refers to the time required for fitness computation and corresponds to the reported times in Table 4.3 and Table 4.5. Specifically, there are three main goals in this section:

- Understand the bottleneck of the monolithic approach by analyzing the percentage of total computation time spent in alignment computation.
- Understand the bottleneck of the recomposition approach by analyzing the percentage of total computation time spent in alignment computation and additional details, e.g., the number of recompositions.
- Compare the bottleneck analysis of both approaches.

The fitness computation between a particular log trace and net requires several other steps such as building a synchronous product net, and constructing the alignments [4]. However, since the majority of the computation time is in the computation of the alignment via the replay of the synchronous product net, previous works often do not always distinguish between the different sources of the total computation time.

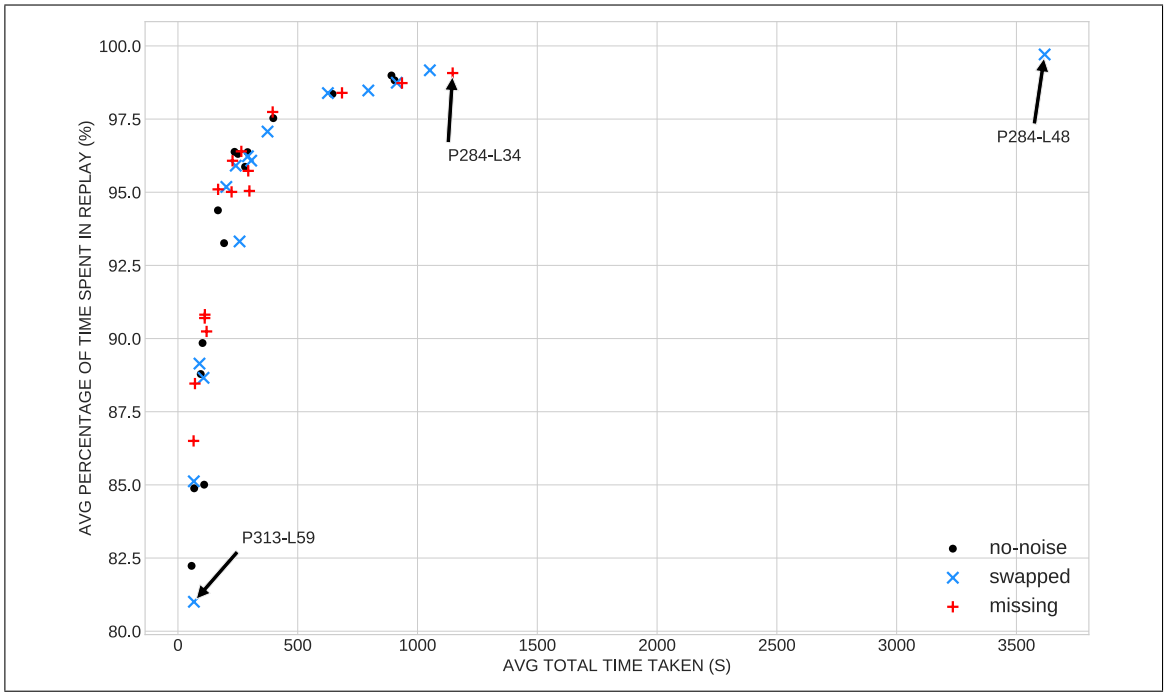


Figure 4.17. Percentage of time spent in alignment computation in relation to total computation time

Figure 4.17 compares the total computation time with the percentage of time spent in replay for all the conducted experiments. It shows the minimum replay percentage as 81% for P313-L59 and the maximum replay percentage of feasible monolithic replays as 99% for P284-L34. The shape of the scatter plot illustrates that the percentage of time spent in replay quickly approximates to 100% as total computation time grows. This confirms that the bottleneck for the monolithic approach is at the replay. Also, this bottleneck exists regardless of the noise type in the data. Next, a similar analysis is conducted for the recomposition approach with several extensions to consider the additional components in the framework.

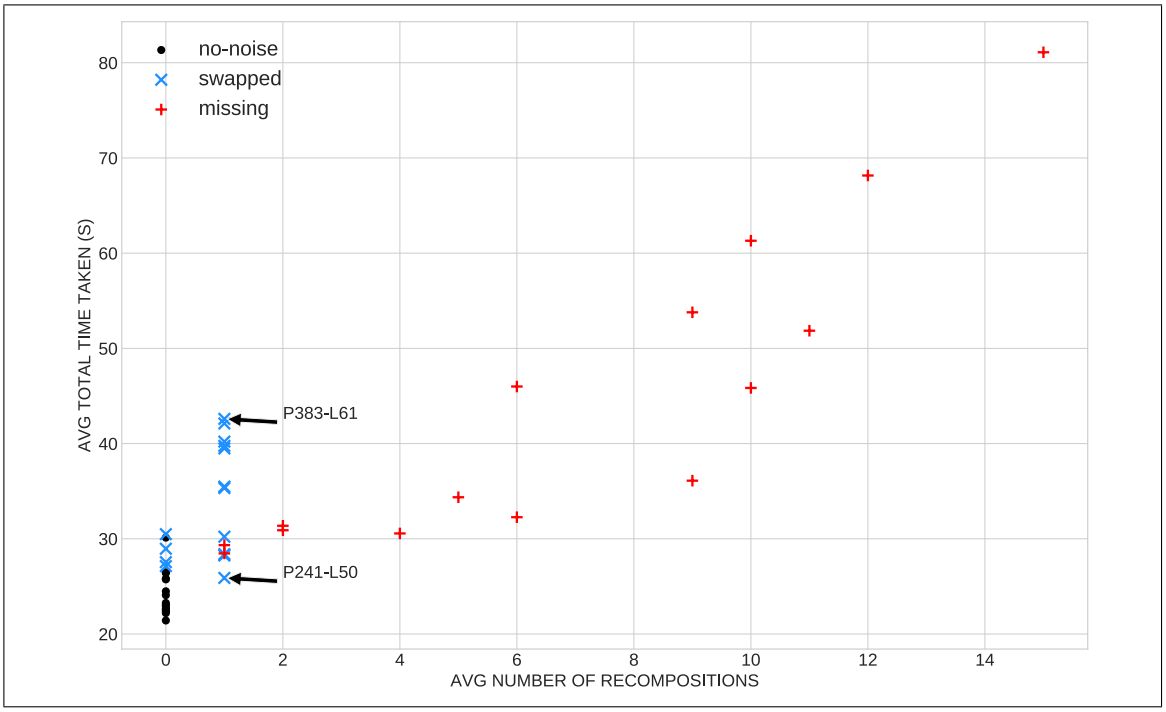


Figure 4.18. Total computation time in relation to number of recompositions for synthetic logs at exact experiments under the recomposition approach

As illustrated by the overview of the recomposition approach in Figure 4.1, if there is no total border agreement between two subalignments, the recomposition approach undertakes a recomposition to merge the corresponding subcomponents before recomputing the alignment in the next iteration. Figure 4.18 compares the number of recompositions with the total computation time for the "exact experiments" under the recomposition approach. In spite of the variances in total computation time at particular recomposition values, it is quite clear that the total computation time increases with the number of recompositions. It is also evident that results of different noise types show different characteristics. We first note that none of the "no-noise" experiments required any recompositions. This is because the subalignments only have synchronous moves so that total border agreement is guaranteed. Contrastingly, this is not the case for the experiments on noisy datasets. For the "swapped" experiments, a subset of the experiments is able to compute the exact fitness value without any recompositions and the rest required exactly one recomposition.

The maximum of one recomposition can be attributed to the fact that only one pair of activities is swapped to create noise during the data generation. This means that if the net is decomposed on the swapped activities then border agreement issue is bound to occur. This is because the subcomponents which have only one of the activities would not be able to detect the activity swap. Since the nets are not maximally decomposed in the experiments, all subcomponents are comprised of more than two unique visible transitions. In the worst case, decomposition would occur on one of the two swapped activities. This means that border agreement issues are immediately resolved when the subcomponents are merged on the decomposed activity. The variances in total computation times for particular recomposition values can be attributed to the underlying data complexity. For example, Figure 4.18 shows that there is a notable variance for the experiments which underwent one recomposition. The minimum time is from P241-L50 and the maximum time is from P383-L61. Referring to Table 4.1, we find that P241 has one of the smallest and simplest structure with 117 activities, 13 ANDs, and 12 XORs. In contrast, P383 has one of the largest and most complex structure with 150 activities, 29 ANDs, and 20 XORs. For the "missing" experiments, the number of recompositions ranges from 1 to 15. Furthermore, they particularly illustrate the direct positive correlation between the two factors.

While it is clear that the total computation time should increase with the number of recompositions, we would like to know whether the proportion of total time spent in replay remains the same as the number of recompositions increases. Figure 4.19 compares the total computation time with the percentage of time spent in replay at the exact experiments under the recomposition approach. Firstly, the figure suggests that the percentage of time spent in replay decreases as total computation time increases. This is a surprising result as it contrasts the previous analysis on the monolithic approach where the percentage of time spent in replay increases with total computation time. Furthermore, the percentage of time spent in replay never exceeds 60% under the recomposition approach whereas the percentage of time spent in replay never falls below 80% under the monolithic approach. This suggests that under the recomposition approach, as computation complexity increases, replay becomes less of a contributing factor to computation time.

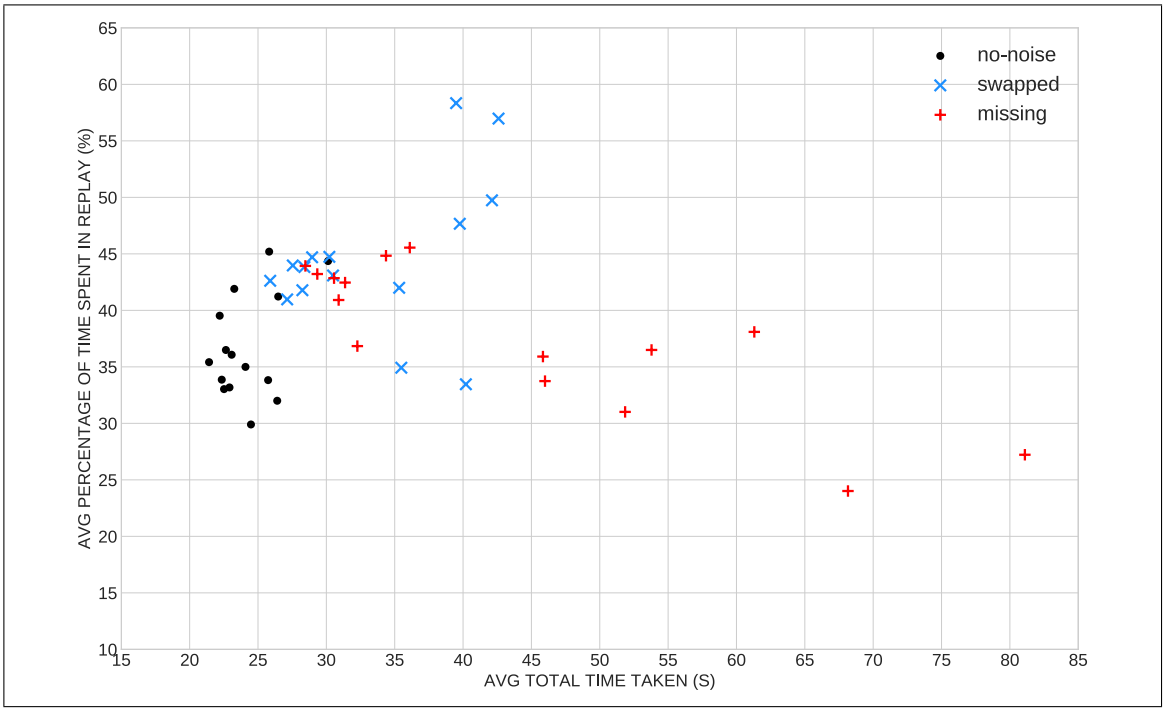


Figure 4.19. Total computation time in relation with to the percentage of time spent in replay for synthetic logs at exact experiments under the recomposition approach

One possible culprit would be the recomposition component of the framework which can be associated to the number of recompositions. Figure 4.20 shows the percentage of time spent in replay in relation to the number of recompositions. Indeed, the figure resembles Figure 4.19 to show that replay has less of an impact on performance as complexity increases. Recall that Figure 4.18 suggested that the number of recompositions and total computation time are positively correlated. Similar to before, there can be variances in the percentage of time spent in replay at particular recomposition values. This can be explained by the number of subnets for each dataset. For example, for experiments that underwent one recomposition, the maximum time percentage is of P275-L52 and the minimum time percentage is of P272-L62. The P275 net has 13 subnets in its initial decomposition and P272 has 32 subnets in its initial decomposition. Having more subnets means more overhead before and after the alignment computation. However, this can likely be alleviated with the use of more computer nodes.

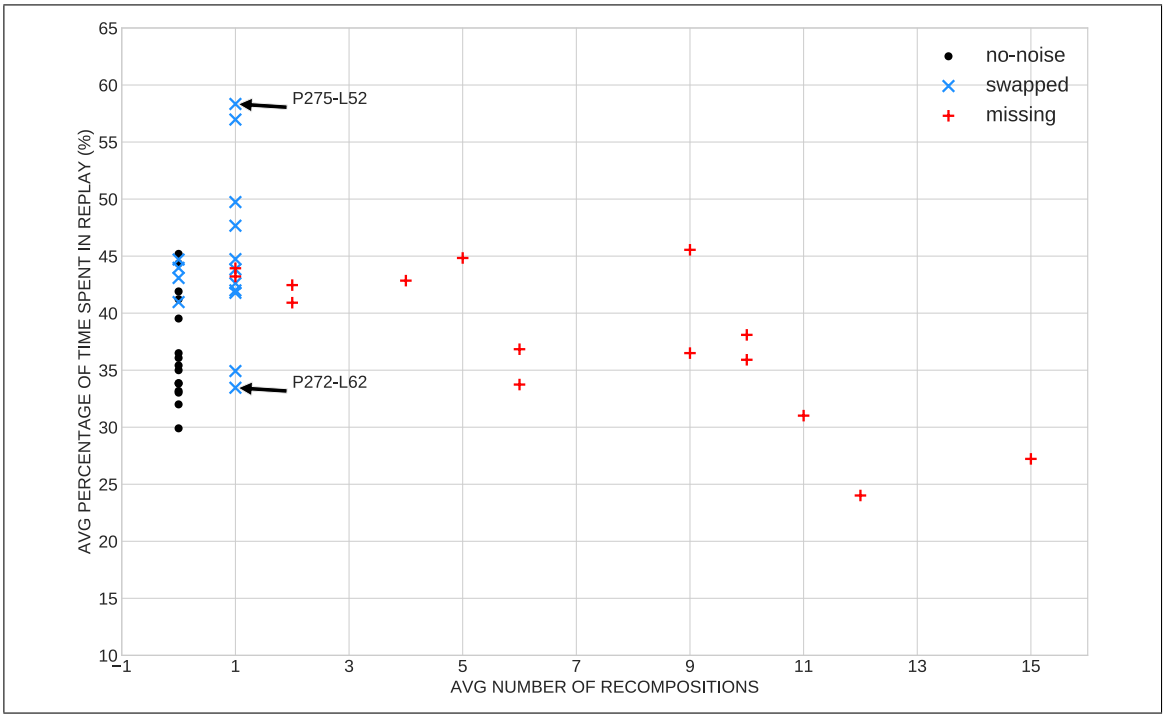


Figure 4.20. Percentage of time spent in replay in relation to number of recompositions for synthetic logs at exact experiments under the recomposition approach

Continuing with this line of reasoning, the low percentage of time spent in replay also prompts investigation into the number of remaining problematic traces that are taken to the next iteration by a recomposition. Figure 4.21 shows the number of remaining trace variants in relation to the number of recompositions for the experiments on the datasets with “missing” noise. We only focus on these experiments because the other experiments were able to align all traces in less than two recompositions. The figure illustrates that for all the experiments, less than 100 trace variants remains to be aligned after the first recomposition. This explains the previous findings which suggested that the percentage of time spent in replay decreases with the number of recompositions. Moreover, it implies that the bottleneck to computing the exact fitness value under the recomposition approach is in having the appropriate recomposition that will permit total border agreement for the remaining traces.

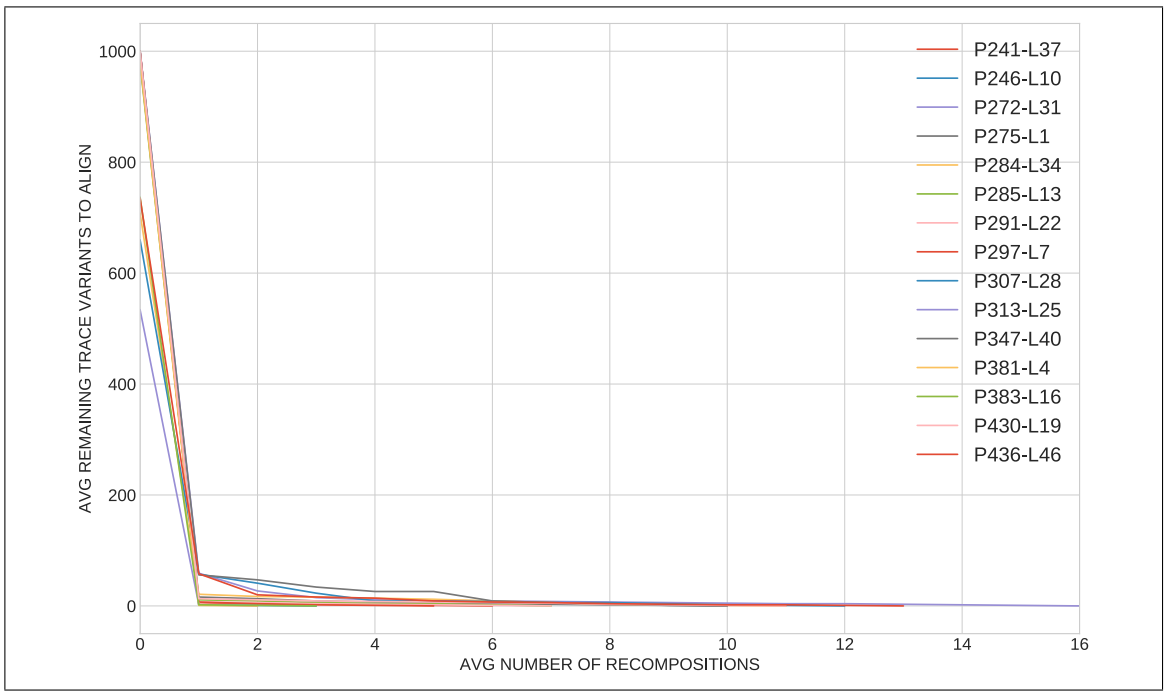


Figure 4.21. Number of remaining trace variants in relation to number of recompositions for the experiments on datasets with “missing” noise

Overall, the analysis on the different aspects of performance leads to several insights. First, we confirm that the bottleneck for the monolithic approach is at the alignment computation. Next, we find that the recomposition approach is able to partially shift the responsibility to the recomposition component. By decomposing the overall component, replay is only performed on small subcomponents even for large and complex processes. This prevents state space explosions during the alignment computation and keeps replay time in check for each subcomponent. It also means that the necessary number of recompositions have a large impact on performance. Moreover, we find that a small number of log traces may require several recompositions before yielding a set of subalignments that fulfills total border agreement. This means that the bottleneck to computing the exact fitness value is in finding the appropriate recomposition that will permit total border agreement. Since the experiments have been performed on only one computer node, distributing the computation across more nodes is likely to lead to further performance gains. The possibility of shifting the bottleneck from replay to recomposition motivates future

work to investigate decomposition and recomposition strategies as well as different decomposed replay approaches. However, we once again emphasize that this shift is partial because the capacity to decompose a system net into small subnets is restricted by whether if the net structure can be decomposed.

4.4.5. Feasibility and interval narrowing time constrained scenarios

One important contribution of the recomposition approach is to enable configurable approximations of the overall conformance between the model and log. This means that it would not be necessary to align all the traces in the log with the model. As previously mentioned, fitness approximations by fitness interval values can further reduce computation times and alleviate replay feasibility problems with specific log traces or sections of the model. As time is often the principal concern for alignment-based conformance checking, this section focuses on the effects of time as a hard constraint on the monolithic and recomposition approaches. There are two main goals in this section:

- To show how to compare the feasibility of the recomposition approach using the monolithic approach. We identify cases where the recomposition approach can provide either an exact or an interval fitness value while monolithic replay is infeasible.
- To analyze the effects on time upon the narrowing.

For the sake of comparison, the experiments are conducted using the same synthetic models and generated logs from the experiments of the previous section. As explained, each model has two associated "noisy" logs; the *no-noise* logs are excluded from these experiments. For each model-log pair, we conduct three experiments, each constrained by a different time limit on the overall alignment time. This is done by varying the value of the GlobalDurationThreshold parameter. Experiments are conducted with the GlobalDurationThreshold values of 1, 5 and 10 minutes.

For the results, in the case of the monolithic approach, an exact fitness value is provided if replay is feasible and it is marked with a cross otherwise. For the recomposition

approach, since it defaults to approximation if an exact value cannot be computed, either an exact fitness value or a fitness interval is given.

Dataset					Time constraint on overall alignment time (min)					
					1'		5'		10'	
Name	$\|A\|$	$\overline{\|\sigma\|}$	$\|L\|$	$\|\{\sigma \in L\}\|$	Monolithic	Recomposition	Monolithic	Recomposition	Monolithic	Recomposition
P241-L37	117	94	1000	1000	✗	0.999	✗	0.999	0.999	0.999
P241-L50	117	95	1000	1000	✗	0.989	✗	0.989	0.989	0.989
P246-L10	137	77	1000	1000	✗	0.998	0.998	0.998	0.998	0.998
P246-L49	137	77	1000	1000	✗	0.987	0.987	0.987	0.987	0.987
P272-L31	201	101	1000	1000	✗	0.972-0.999	✗	0.999	✗	0.999
P272-L62	201	101	1000	1000	✗	0.989	✗	0.989	✗	0.989
P275-L1	101	73	1000	984	✗	0.998	0.998	0.998	0.998	0.998
P275-L52	101	74	1000	965	✗	0.984	✗	0.984	✗	0.984
P284-L34	170	106	1000	708	✗	0.990-0.998	✗	0.998	✗	0.998
P284-L48	170	106	1000	589	✗	0.997	✗	0.997	✗	0.997
P285-L13	140	46	1000	978	✗	0.997	0.997	0.997	0.997	0.997
P285-L56	140	48	1000	973	✗	0.947	0.947	0.947	0.947	0.947
P291-L22	170	90	1000	1000	✗	0.999	✗	0.999	✗	0.999
P291-L51	170	91	1000	1000	✗	0.989	✗	0.989	✗	0.989
P297-L7	125	59	1000	1000	✗	0.998	0.998	0.998	0.998	0.998
P297-L55	125	60	1000	1000	✗	0.982	0.982	0.982	0.982	0.982
P307-L28	193	22	1000	660	✗	0.994	0.994	0.994	0.994	0.994
P307-L53	193	23	1000	581	✗	0.996	0.996	0.996	0.996	0.996
P313-L25	185	20	1000	534	✗	0.993	0.993	0.993	0.993	0.993
P313-L59	185	20	1000	465	✗	0.986	0.986	0.986	0.986	0.986
P347-L40	212	60	1000	1000	✗	0.998	0.998	0.998	0.998	0.998
P347-L58	212	59	1000	1000	✗	0.996	0.996	0.996	0.996	0.996
P381-L4	111	73	1000	983	✗	0.998	0.998	0.998	0.998	0.998
P381-L63	111	74	1000	973	✗	0.985	0.985	0.985	0.985	0.985
P383-L16	150	106	1000	737	✗	0.999	0.999	0.999	0.999	0.999
P383-L61	150	111	1000	632	✗	0.979	✗	0.979	✗	0.979
P430-L19	160	104	1000	1000	✗	0.999	0.999	0.999	0.999	0.999
P430-L57	160	104	1000	1000	✗	0.997	✗	0.997	0.997	0.997
P436-L46	230	35	1000	735	✗	0.983-0.996	0.996	0.996	0.995	0.996
P436-L54	230	35	1000	633	✗	0.982	0.982	0.982	0.982	0.982

Table 4.6. Time-constrained conformance analysis on synthetic logs with noise of dataset using manual initial decomposition

Table 4.6 shows the results of the experiments. For the experiments with a 1 minute time constraint, monolithic replay is not feasible for any dataset. The recomposition approach can compute an exact fitness value for 27 model-log pairs and fitness interval values for the remaining 3 model-log pairs. For the fitness interval results, the interval range varies from 0.008 (P284-L34) to 0.027 (P272-L31). For the experiments with a 5 minute time constraint, monolithic replay is feasible for 19 model-log pairs. The recomposition approach can compute an exact fitness value for all 30 model-log pairs. For the experiments with a 10 minute time constraint, monolithic replay is feasible for 22 model-log

pairs. The recomposition approach can compute an exact fitness value for all 30 model-log pairs. Previous experiments (cf. Section 4.4.3) have shown that the replay of 1 model-log pair cannot be finished within one hour under the monolithic approach.

In terms of feasibility, the results show that the recomposition approach can give an exact fitness value for more model-log pairs than the monolithic approach under all three time constraints. For the shortest 1 minute constraint, the recomposition approach can give an exact fitness value for almost all of the datasets while monolithic replay was not feasible for any dataset. Other than feasibility, the fitness interval results computed by the recomposition approach show that the exact fitness values (as shown in Table 4.5) are always within the interval.

In conclusion, the recomposition approach outperforms the monolithic approach in feasibility when there is a constraint on alignment time. More importantly, the recomposition approach can always give a fitness result to reflect the conformance level between the model and log whereas the monolithic approach can only give a result if replay can be fully completed.

Having compared the performance of the recomposition approach with the existing monolithic approach for different process characteristics and noise scenario, we show that the recomposition approach can be applied to real-life datasets.

4.4.6. Recomposed fitness in real-life cases

Dataset								Monolithic			Recomposition					
Name	$\|A\|$	$\overline{\|\sigma\|}$	$\|L\|$	$\|\{\sigma \in L\}\|$	|AND|	|XOR|	|Loop|	Feasible	Fitness	Time (s)	Feasible	Fitness	Time (s)	$\overline{\text{Steps}}$	$\overline{\text{Speedup}}$	$\widetilde{\text{Speedup}}$
BPIC2012	36	20	13087	4366	4	19	7	✗		> 3600	✓	0.647-0.648	1761	11	> 2.0	> 2.0
BPIC2017	40	24	31509	7478	1	13	3	✓	0.992	71	✓	0.992	78	9	(1.1)	(1.1)

Table 4.7. Replay feasibility and computation times for BPIC2012

In this section we apply the monolithic approach and the recomposition approach on two real-life datasets. This is to demonstrate that the recomposition approach can be applied to real-life datasets and to compare the performances of both approaches. Notice

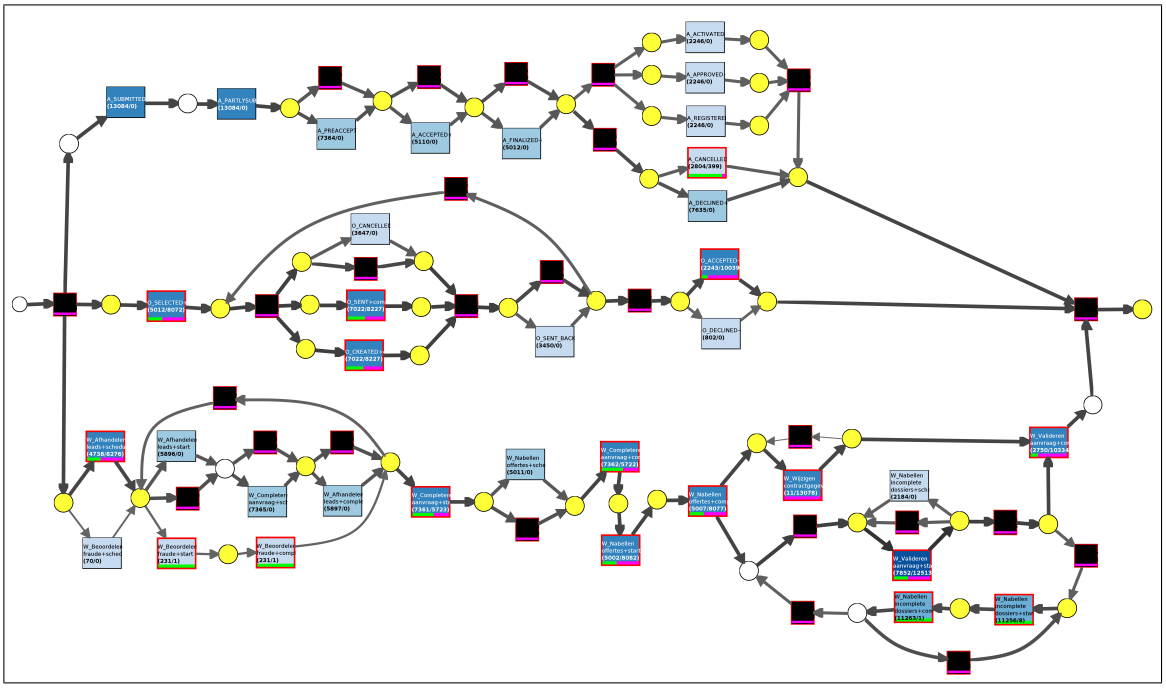


Figure 4.22. Handmade model for the BPIC2012 real-life dataset projected with the deviation issues between the model and log

that the final goal of any conformance approach is to gain insight on the conformance problems. Therefore, we perform a simple analysis using the conformance results from replay to showcase the applicability of alignment-based conformance checking for the analysis of deviations.

We used the BPIC2012 real-life dataset [72] adopted in 2012 for the BPI (Busines Process Intelligence) Challenge. This dataset is taken from a Dutch Financial Institute and contains 13,087 cases and 36 event classes. Apart from some anonymization, the log contains all data as it came from the financial institute.

The process represented in the event log is an application process for a personal loan or overdraft within a global financing organization. The event log is a merger of three intertwined sub processes such that the first letter of each task can be used to identify the subprocess that the task originated from.

Since no explicit process model is provided with the dataset, a model is constructed with consideration to the task semantics and the three sub processes as illustrated in Figure 4.22 (please ignore the color bars at the bottom of transitions and the color of transitions and places for now).

For the experiments, a 1 hour time limit is set such that replays which are still running after the time limit are stopped and deemed infeasible. The recomposition approach is initiated with a maximal decomposition of the model and log. A maximal decomposition is a valid decomposition where the subcomponents are as small as possible. We report the time and fitness of replay under both approaches.

As shown in Table 4.7, we find that the monolithic approach is not feasible for BPIC2012 under the 1 hour time limit. In contrast, the recomposition approach was able to provide a fitness interval value with a range of less than 0.001 under 30 minutes (1800 seconds). The results also show that an interval value was given rather than an exact value because 1 trace was rejected during the replay process given the exclusion parameters defined (cf. Section 4.3). The fitness interval of $[0.647 - 0.648]$ indicates there are considerable deviations between the modeled behavior and observed behavior. Further analysis can be done by visualizing the conformance results.

Conformance results can be visualized through alignments as shown in Figure 4.8 or by projecting the deviation issues onto the corresponding places and transitions as shown in Figure 4.22. Transitions are augmented with two additional information. Firstly, the color of the visible transitions indicates the execution frequency of a particular transition in the log. A lighter color means that the transition is rarely executed and a darker color means that the transition is often executed. The color at the bottom of transitions indicates the distribution between synchronous moves (light green) and model moves (dark pink). A color bar with a high green portion means that there is little conformance issue with the corresponding transition and a low green portion means there is severe conformance issue with the transition. Transitions without color bars do not have any conformance issues.

Log moves are shown by marking the places where log moves occurred. The occurrence frequency is indicated by the size of the places.

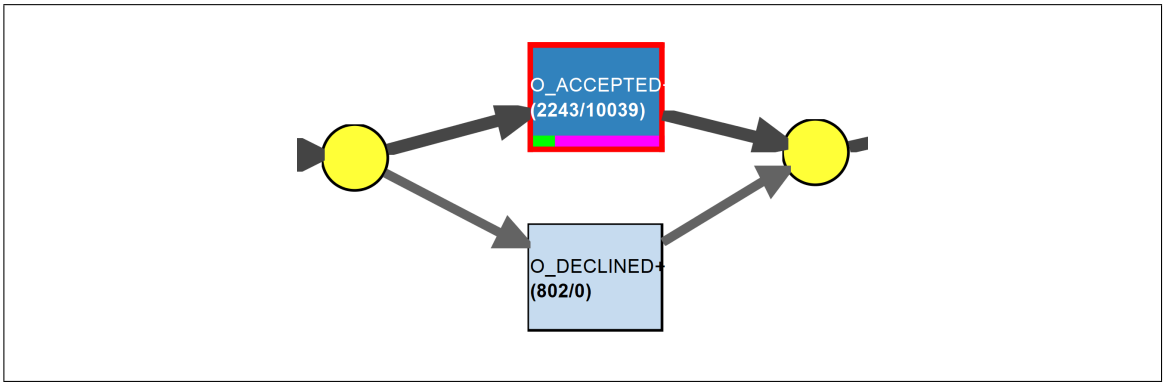


Figure 4.23. Conformance diagnosis on transition *O_ACCEPTED*

Figure 4.24. Alignment for case 173733 with model move on transition *O_ACCEPTED* (highlighted in white)

We first observe that out of the three subprocesses, subprocess *A* which relates to the application itself has minimal conformance issues. Since the application is submitted through the webpage, it makes sense that the observed behavior can be easily and accurately described by a corresponding model. Contrastingly, both subprocess *O* which relates to the offer of the application and subprocess *W* which relates to the work items belonging to the application have major conformance issues. For simplicity, we only focus on subprocess *O*. There are log moves relating to all parts of the subprocess and there is a high frequency of model moves relating to the transitions: *O_SELECTED*, *O_SENT*, *O_CREATED* and *O_ACCEPTED*. For example, figure 4.23 shows the ratio between synchronous moves (at 2243 cases) and model moves (at 10039 cases) for transition *O_ACCEPTED*. This means that 10039 cases did not execute transition *O_ACCEPTED* even though they were supposed to according to the model. This can be confirmed

by inspecting the alignments of particular cases in the alignment visualization. Figure 4.24 shows the alignment for case 173733 which indicates a model move for transition *O_ACCEPTED*. As such, there is a discrepancy between the reality and the model suggested by the task semantics regarding to how an offer is processed. The identification of this conformance issue may prompt further investigations with related stakeholders such as the employees who handled the deviating cases.

In the previous example, replay is not feasible under the monolithic approach. Here, we further showcase an example using a real-life dataset where replay is feasible under both approaches.

We used the BPIC2017 real-life dataset [73] adopted in 2017 for the BPI (Business Process Intelligence) Challenge. This dataset is taken from the same Dutch Financial Institute and contains 31,509 cases and 66 event classes. The data contains all the applications filed through the online system in 2016 and their subsequent events until the 1st of February 2017, 15:11. Note that the data is provided provided by the same company of the BPIC2012 dataset but has underwent several changes and improvements to the system. One important difference is that now the system can support multiple offers for a single application.

In spite of the changes, the process is still a merger of three intertwined subprocesses such that the first letter of each task can be used to identify the subprocess that the task originated from. To ensure the replay feasibility of both approaches, the event log is filtered to keep only events with a schedule, start, and complete lifecycle attribute value. The filtered log has 40 event classes.

Since no explicit process model is provided with the dataset, a model is discovered using the inductive miner as illustrated in Figure 4.25 (please ignore the color bars at the bottom of transitions and the color of transitions and places for now).

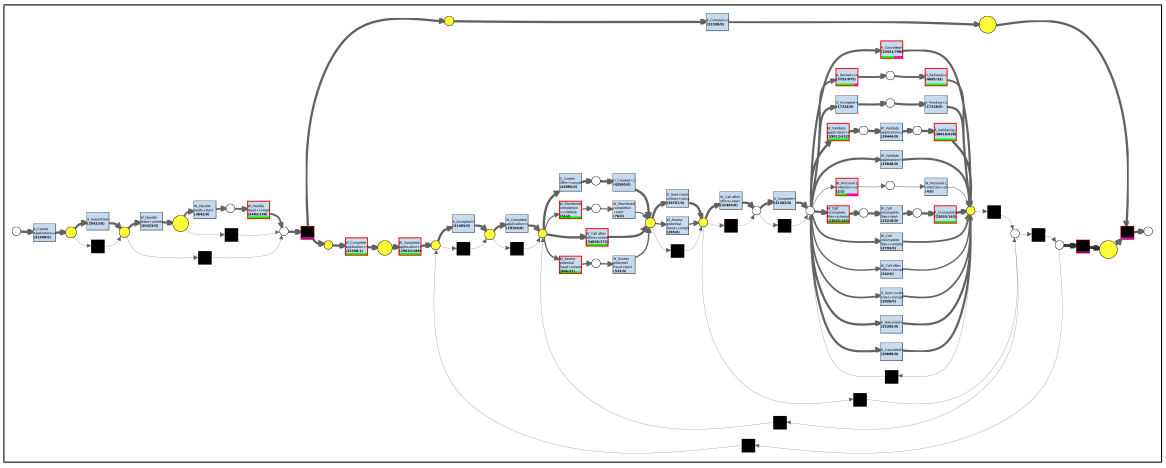


Figure 4.25. Discovered model for the BPIC2017 real-life dataset projected with the deviation issues between the model and log

Figure 4.26. Alignment for case Application_931736025 with a model move on transition *A_Cancelled+complete* (highlighted in white)

The setup for the experiments is the same as the previous real-life dataset example with the same initial decomposition strategy for the recomposition approach and time limit for both approaches.

As shown in Table 4.7, replay was feasible under both approaches. Both approaches finished computing the model fitness in less than 1.5 minutes with the monolithic approach being 7 seconds faster on average so that there is an average (and median) speedup of 1.1 under the monolithic approach. This shows that even for simpler models, the recomposition approach is only slightly slower despite the additional recomposition procedures.

Both approaches yielded a fitness value of 0.992 which shows that the discovered model can describe nearly all the observed behavior in the log. Other than having the same fitness value, the computed alignments under both approaches are composed of the same alignment moves. This means that both approaches lead to the same enriched model in Figure 4.25 when the alignments are projected onto the model. Furthermore, examining

most of the individual trace alignments finds that the alignment for the same case to be the same under both approaches. For example, Figure 4.26 shows the alignment for case Application_931736025. The alignment computed under both approaches indicates a model move on activity *A_Cancelled+complete*. As previously mentioned, given a cost function, there can be multiple optimal alignments for a particular log trace. Further investigations into the circumstances under which the monolithic and recomposition approaches compute different optimal alignments can be interesting, especially if there are reasons to prefer a particular one.

Lastly, this example demonstrates the need to consider the other quality dimensions for the evaluation of model quality. While the high fitness suggests that this is a "good" model, a closer look at the model by a user would conclude otherwise. For example, there are multiple loop constructs so that each loop construct approximates a flower model that can generate any sequences involving the transitions in the loop. Figure 4.27 illustrates one of these loop constructs. The construct initiates with an XOR split of 12 branches. Then, at the output place where all the branches terminate, there is an invisible transition that loops back to the previous XOR split. This means that if a case reaches a marking that includes the place of the XOR split, the model would then be able to replay any sequences that involve one or more of the branches. As such, this part of the model gives little information on the control flow of the observed behavior.

In conclusion, the results clearly demonstrate that the applicability of the recomposition approach make alignment-based conformance checking feasible for real-life datasets. In addition, for the example where replay is feasible under both approaches, an analysis of the computed alignments confirm that not only fitness values match but also alignments show the same behavioral analysis results. Lastly, the utility of the conformance results is shown through a simple analysis.

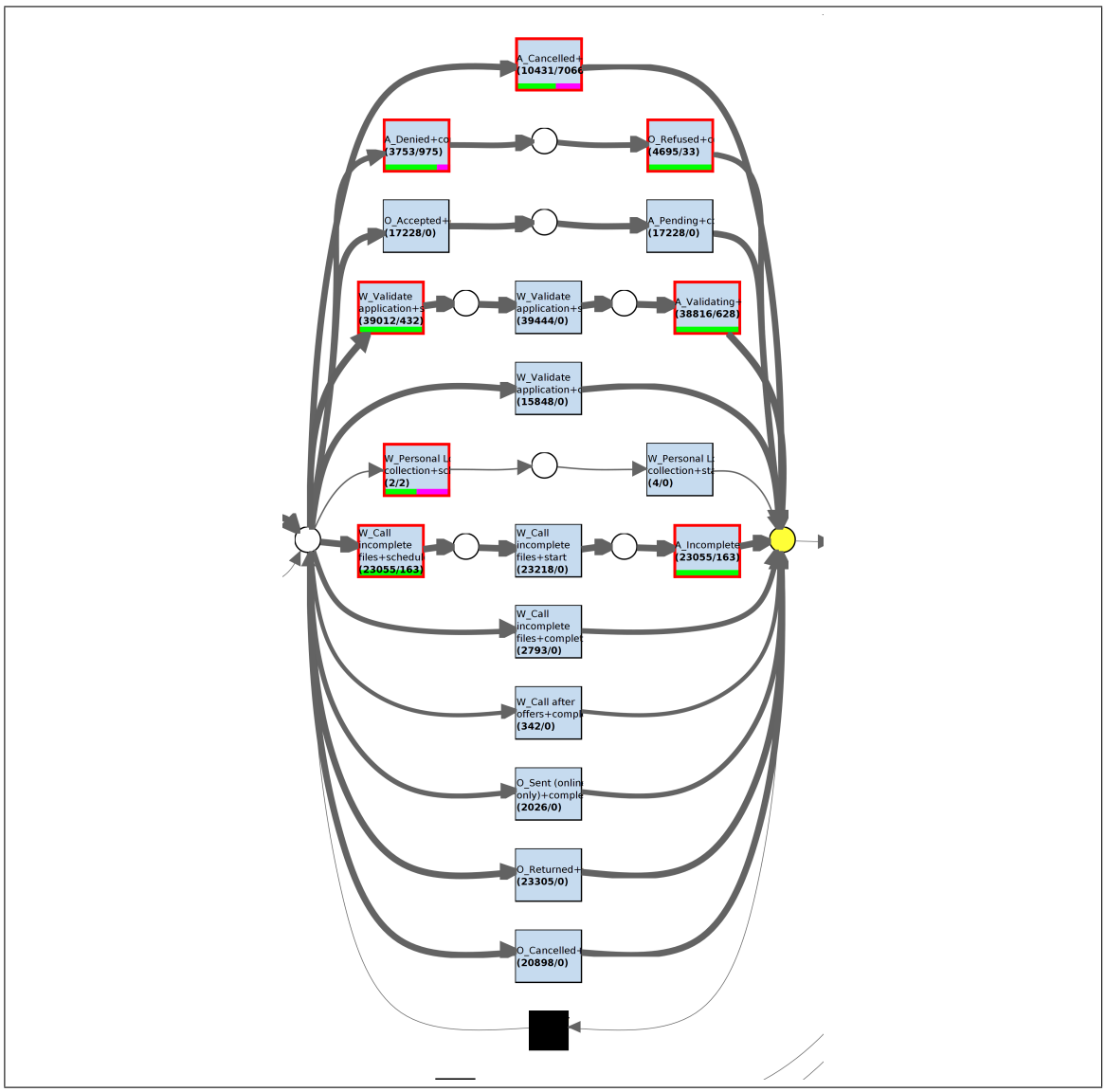


Figure 4.27. Loop construct in the discovered model of the BPIC2017 real-life dataset projected with deviation issues

4.5. Related work

Several approaches have been proposed to decompose conformance checking in the literature [89, 65, 91, 50]. Their experimental results showed an immense reduction in computation time over the existing monolithic approach. However, conformance results

from most of these approaches only remain at the level of sub-logs and sub-nets, the conformance between the overall process model and event log is not computed. For example, a recent paper [91] proposed a workflow decomposition technique that decomposes a workflow net into a set of smaller workflow nets by places after which conformance checking is performed on the sub-nets by alignment. This treats sub-components independently and does not compute alignment at a global level. An exception is the implementation of an earlier proposed generic approach to decompose process discovery and conformance checking [89]. In the implementation, conformance checking is again performed by aligning sub-logs and sub-nets but an extra step is taken afterwards to merge the sub-alignments into an overall pseudo alignment. The pseudo alignment gives a lower bound for the mismatch costs between the overall process model and log. This guarantee can provide an idea about the overall conformance. In relation to these works, the proposed recomposition approach advances existing decomposition solutions by extending decomposition techniques to compute an exact or interval value that reflects the overall conformance between a process model and log.

Other than decomposing conformance checking, other ideas have also been proposed. In [61], approximate alignments are proposed to view sections of step-moves in an alignment under a coarser granularity as multisets. This abstract view of alignment reduces the complexity of the alignment problem so that computation time is reduced. Another way to reduce the complexity of the problem is through reduction of the model and log. In [62], the notion of indication is used to reduce process models. The occurrence of an indicator transition requires the presence of a specified set of transitions (its indicated set) due to their causal relations. As such, these indication relations between transitions can be used to reduce models and logs. By using a reduced model and log, a macro-alignment can be computed in significantly shorter time. The missing alignment details are later expanded using an efficient alignment algorithm from bio-informatics. Other than abstracting alignment details to alleviate complexity, new interpretations of the alignment concept have

also been proposed such as anti-alignments [18]. Compared to these alternatives, the recomposition approach presented in this chapter preserves the alignment concept and does not need to abstract away details to achieve performance gain.

Another related work is conformance checking of proclets [25]. Proclets can be used to define so-called artifact centric processes, i.e., processes that are not monolithic but are composed of smaller interacting processes (called proclets). In [25], it is shown that conformance checking can be done per proclet by projecting the event log onto a single proclet while considering interface transitions in the surrounding proclets.

4.6. Conclusions

This chapter presents the recomposing conformance checking framework that computes optimal alignments in a divide and conquer manner. This provides a way of computing alignments for large and complex models where monolithic approaches are not suitable. The approach iteratively solves decomposed alignment problems and merges on border activities to resolve border conflicts.

To enable the aggregation of results from subcomponents, we make use of the total border agreement condition presented in Chapter 3. In addition, we adapted the existing relative fitness metric as the decomposed fitness metric to compute an exact or interval value to reflect the fitness between a model and log.

The conformance checking approach has been implemented in ProM [37]. Extensive experimental results from synthetic and real-life datasets demonstrate the potential performance gains of the recomposition approach over the existing state of the art monolithic approach. The use of decomposition increased replay feasibility and reduced computation time for datasets feasible under both approaches. The speedup from adopting the recomposition approach has shown to be attainable under different noise scenarios and increases with the average trace lengths of the event logs.

In the experiments, we found that the initial decomposition for instantiating recomposing replay can have an impact on its performance. For the sake of simplicity, manual decompositions were used in the experiments on the synthetic datasets. We concede that using a manual decomposition is a possible weakness of the approach, but believe that we can generate such decompositions in an automated way by using SESE techniques. Also, analyzing the performance of the recomposition approach, we found that the bottleneck of the proposed approach is in not being able to reach the required total border agreement condition for some traces so that these traces can take many iterations to align. In the next chapter, we present our investigation into different heuristics that aim to encourage the border condition so that this bottleneck can be alleviated.

5. Improving merging conditions for recomposing conformance checking

5.1. Introduction

As discussed in the previous chapter, our experimental results found that for the proposed recomposing conformance checking framework, the bottleneck is in that some log traces require many iterations to reach the needed merging condition. In this chapter, we present various heuristics that can be applied at the recomposition step. We also present experimental results on both synthetic and real life data that show significant performance gains in terms of computation time when the proposed heuristics are applied.

The rest of the chapter is organized as follows: Section 5.2 presents a running example used in the chapter. Section 5.3 presents the recomposition approach as the focus. Section 5.4 defines and structures the recomposition step and Section 5.5 sheds light on the limitations of the existing recomposition strategies. Section 5.6 presents four recomposition strategies that can be used in the recomposition step. Section 5.7 details the experimental setup for the evaluation of the proposed strategies, and Section 5.8 analyzes the experimental results. Section 5.9 presents the related work. Finally, Section 5.10 presents some conclusions.

5.2. Running example

Figure 5.1 presents a system net S that models a loan application process for property purchase (ignore the grey boxes in the background for now). $[i]$ is the initial marking and $[o]$ is the final marking. For example, an instance of the process might initiate with the receipt of a loan application a, then the applicant's credit history would be checked b, his/her loan risk assessed c, and the value of the property appraised d. The results are then consolidated for the assessment of the applicant's eligibility f. Afterwards, if the application is successful, an acceptance pack is then prepared g and sent h to the applicant. At some point in the future, the repayment is verified i and the application is then archived

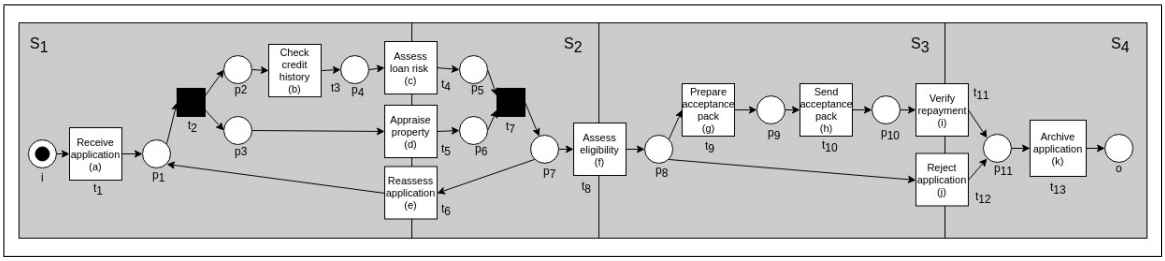


Figure 5.1. System net S that models a loan application process

$$L = [\overbrace{\langle a, b, c, d, f, g, h, i, k\rangle}^{\sigma_1}{}^{5}, \overbrace{\langle a, c, b, d, f, g, i, h, k\rangle}^{\sigma_2}{}^{10}, \overbrace{\langle a, c, b, d, f, g, j, k\rangle}^{\sigma_3}{}^{5}]$$

Figure 5.2. Running example: Event log L

k as the process is terminated. The sequence of occurred events can be represented as the activity sequence $\langle a, b, c, d, f, g, h, i, k\rangle$. In real-life, the process executions are recorded as event data and can be expressed as an event log.

Figure 5.2 presents an event log L corresponding to the system net in Figure 5.1. Log L has 20 cases in total with 5 cases following trace σ_1, 10 cases following trace σ_2, and 5 cases following trace σ_3. In cost-based alignment conformance checking, a trace is aligned with the corresponding system net to produce an alignment.

Figure 5.1 presents a valid decomposition D of net S where sub-nets are marked by the grey boxes. For example, sub-net S_1 consists of the transitions t_1, t_2, t_3, t_4, t_5, and t_6. Border activities can be identified as the activities of the transitions that are shared between two sub-nets. They are t_4, t_5, t_6, t_8, t_{11}, and t_{12}. Under the recomposition approach framework, overall alignments can be computed in a decomposed manner.

5.3. Recomposing conformance checking

Figure 5.3 presents an overview of the recomposing conformance checking framework [38, 37] which consists of the following five steps:

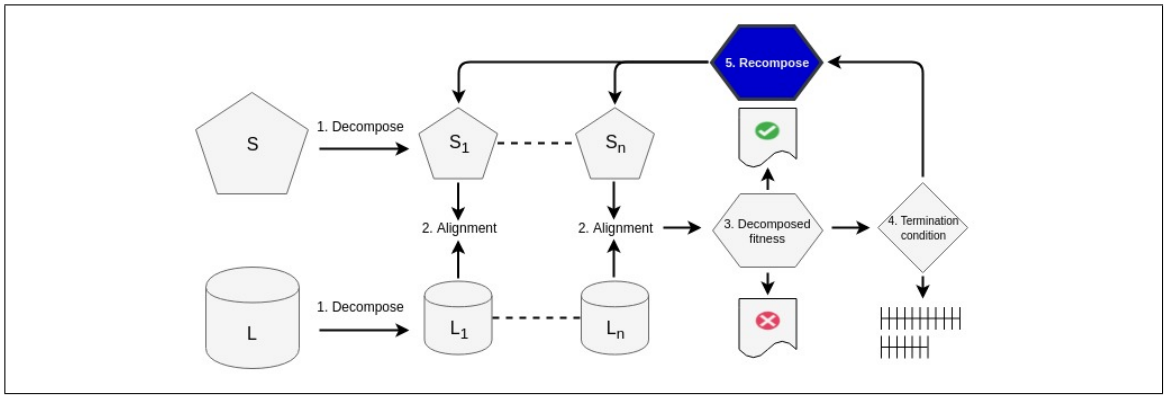


Figure 5.3. Recomposing conformance checking framework with the recomposition step highlighted in dark blue

(i) The net and log are decomposed using a decomposition strategy, e.g., maximal decomposition [65].

(ii) Alignment-based conformance checking is performed per sub-net and sub-log to produce a set of sub-alignments for each log trace.

(iii) Since sub-components overlap on border activities, the set of sub-alignments for each log trace also overlap on moves involving border activities. In [38], it was shown that if the sub-alignments synchronize on these moves, then they can be merged as an overall optimal alignment using the merging algorithm presented in [88]. This condition was formalized as the *total border agreement* condition. Log traces that do not meet the requirement are either rejected or left for the next iteration. As such, only border activities can cause merge conflicts.

(iv) User-configured termination conditions are checked at the end of each iteration. If the framework is terminated before computing the overall optimal alignments for all log traces, then an approximate overall result is given. The results of the framework consist of a fitness value and a set of alignments corresponding to the log traces. In the case of an approximate result, the fitness value would be an interval bounding the exact fitness value and the set of alignments would have pseudo alignments.

(v) If there are remaining log traces to be aligned and the termination conditions are not reached, then a recomposition step is taken to produce a new net decomposition and

a corresponding set of sub-logs. The next iteration of the framework then starts from Step (2).

While experimental results have shown significant performance gains from the recomposition approach over its monolithic counterpart, large scale experimentation has shown that recomposition is a potential bottleneck. In particular, the strategies used at the recomposition step can have a significant impact. The following section takes a more detailed look at the recomposition step and discusses the limitations of the current recomposition strategies.

5.4. Recomposition step

The recomposition step refers to Step 5 of the framework overview presented in Figure 5.3 and is highlighted in dark blue. We formalize the step in two parts: the production of a new net decomposition and a corresponding set of sub-logs.

Definition 5.1 (Recomposition step)**.** *Let $D \in \mathcal{D}(S)$ be a valid decomposition of system net S and let $L = \mathcal{B}(A^*)$ be an event log. For $1 \leq i \leq n$, where $n = |D|$, let $M_i = (A_i \cup \{\gg\}) \times (T_i \cup \{\gg\})$ be the possible alignment moves for a sub-component so that $\Gamma_D = [(\gamma_{i_1}, \ldots, \gamma_{i_n}) \in M_1^* \times \ldots \times M_n^* \mid \exists_{\sigma_i \in L} \forall_{j \in \{1,\ldots,n\}} \pi_1(\gamma_{i_j}) \restriction_{A_j} = \sigma_i \restriction_{A_j}]$ contains the latest sub-alignments for all log traces.*

Given the valid decomposition, and the latest sub-alignments, $R_S : \mathcal{D}(S) \times \mathcal{B}(M_1^* \times \ldots \times M_n^*) \to \mathcal{D}(S)$ creates a new valid decomposition $D' \in \mathcal{D}(S)$ where $m = |D'| < |D|$. Then, given the new and current net decompositions, the event log, and the latest sub-alignments, $R_L : \mathcal{D}(S) \times \mathcal{D}(S) \times \mathcal{B}(A^*) \times \mathcal{B}(M_1^* \times \ldots \times M_n^*) \nrightarrow \mathcal{B}({A'_1}^*) \times \ldots \times \mathcal{B}({A'_m}^*)$ creates a set of sub-logs to align in the following iteration of the recomposition approach[1].

[1]Note that this is a partial function because some tuples of sub-logs will not have a corresponding element in the domain. For example, valid decomposition requires sub-traces to overlap on border activities, a tuple of sub-logs that have sub-traces with events involving certain border activities in one sub-trace but none in another would be unreachable from the domain, e.g., $([\langle a, b, c\rangle], [\langle f\rangle], [\langle f, g, h, i\rangle], [\langle i, k, \rangle])$

Overall, the recomposition step R creates a new net decomposition and a corresponding set of sub-logs, $R : \mathcal{D}(S) \times \mathcal{B}(A^*) \times \mathcal{B}(M_1^* \times \ldots \times M_n^*) \nrightarrow \mathcal{D}(S) \times \mathcal{B}({A'_1}^*) \times \ldots \times \mathcal{B}({A'_m}^*)$.

The current recomposition strategy involves recomposing on the most frequent conflicting activities (MFC) and constructing sub-logs that contains to-be-aligned traces which carry conflicting activities that have been recomposed upon (IC).

Most frequent conflict (MFC) recomposes the current net decomposition on the activity set $A_r = \{a \in A_b(D) \mid a \in \arg\max_{a' \in A_b(D)} \sum_{\boldsymbol{\gamma_i} \in \text{Supp}(\Gamma_D)} C(\boldsymbol{\gamma_i})(a')\}$ where $\Gamma_D \in \mathcal{B}(M_1^* \times \ldots \times M_n^*)$ are the latest sub-alignments and $C : M_1^* \times \ldots \times M_n^* \to \mathcal{B}(A)$ is a function that gives the multiset of conflicting activities of sub-alignments. Hence, A_r contains the border activities with the most conflicts.

Inclusion by conflict (IC) then creates a log $L_r = [\sigma_i \in L \mid \exists_{a \in A_b(D)}\ C(\boldsymbol{\gamma_i})(a) > 0 \wedge a \in A_r]$ where $\boldsymbol{\gamma_i} \in \Gamma_D$ are the sub-alignments of trace $\sigma_i \in L$ and net decomposition $D \in \mathcal{D}(S)$. As such, log L_r includes to-be-aligned log traces which have conflicts on at least one of the border activities that have been recomposed upon. Later, log L_r is then projected onto the new net decomposition to create the corresponding sub-logs.

5.5. Limitations to the current recomposition strategies

To explain the limitations, we refer to the set of optimal sub-alignments in Figure 5.4 from aligning net decomposition D in Figure 5.1 and log L in Figure 5.2. We first note that for the conflicting activities which are highlighted in grey: $\sum_{\boldsymbol{\gamma} \in \Gamma_D} C(\boldsymbol{\gamma})(c) = 2$, $\sum_{\boldsymbol{\gamma} \in \Gamma_D} C(\boldsymbol{\gamma})(i) = 1$, and $\sum_{\boldsymbol{\gamma} \in \Gamma_D} C(\boldsymbol{\gamma})(j) = 1$, where $\Gamma_D = \{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \boldsymbol{\gamma}_3\}$. With activity c being the most frequent conflicting activity, MFC recomposes the current net decomposition on $A_r = \{c\}$ and IC creates the corresponding sub-logs containing $L_r = \{\sigma_2, \sigma_3\}$ since both traces have activity c as a conflicting activity. The new net decomposition will contain three sub-nets rather than four where sub-net S_1 and sub-net S_2 are recomposed upon activity c as a single sub-net. The corresponding sub-log set is created by projecting log L_r onto the new net decomposition.

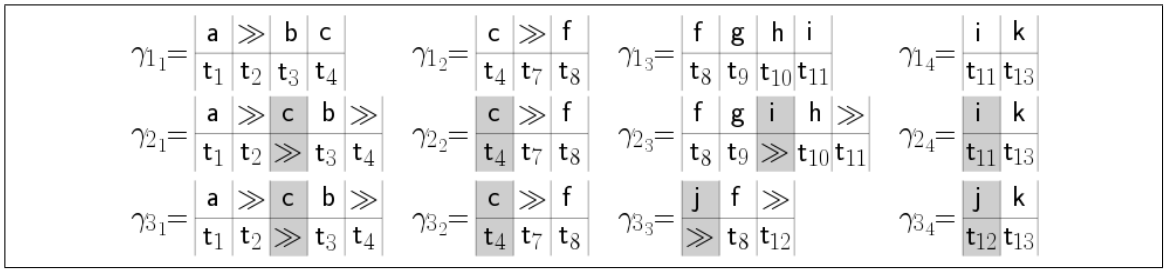


Figure 5.4. Sub-alignments $\boldsymbol{\gamma_1} = (\gamma_{1_1}, \gamma_{1_2}, \gamma_{1_3}, \gamma_{1_4})$, $\boldsymbol{\gamma_2} = (\gamma_{2_1}, \gamma_{2_2}, \gamma_{2_3}, \gamma_{2_4})$, and $\boldsymbol{\gamma_3} = (\gamma_{3_1}, \gamma_{3_2}, \gamma_{3_3}, \gamma_{3_4})$ of log L_1 and net decomposition D_1 with merge conflicts highlighted in grey

While one merge conflict is resolved by recomposing on activity c, the merge conflicts at activity i and j will remain in the following iteration. In fact, under the current recomposition strategy, trace σ_2 and σ_3 have to be aligned three times each to reach the required merging condition to yield overall alignments. This shows the limitation of MFC in only partially resolving merge conflicts on the trace level and IC in leniently including to-be-aligned log traces whose subsequent sub-alignments are unlikely to reach the necessary merging condition.

As such, the key to improving the existing recomposition strategies is in lifting conflict resolution from the individual activity level to the trace level so that the net recomposition strategy resolves merge conflicts of traces rather than activities and the log recomposition strategy selects log traces whose merge conflicts are likely to be fully resolved with the latest net recomposition. In the following section, three net recomposition strategies and one log recomposition strategy are presented. These strategies improve on the existing ones by looking at merge conflict sets, identifying co-occurring conflicting activities, and minimizing the average size of the resulting recomposed sub-nets. The later experimental results show that the strategies lead to significant performance improvements in both synthetic and real-life datasets.

5.6. Recomposition strategies

In this section, three net recomposition strategies and one log recomposition strategy are presented.

5.6.1. Net recomposition strategies

As previously shown, resolving individual conflicting activities may only partially resolve the merge conflicts of traces. This key observation motivates the following net recomposition strategies which target conflicts at the trace level.

Top k most frequent conflict set (MFCS-k) constructs a multiset of conflict sets $A_{cs} = [\text{Supp}(C(\boldsymbol{\gamma})) \subseteq A_b(D) \mid \boldsymbol{\gamma} \in \Gamma_D \wedge |C(\boldsymbol{\gamma})| > 0]$. Then the top k most frequent conflict set $A_{cs,k} \subseteq \{a_{cs} \subseteq A_b(D) | A_{cs}(a_{cs}) > 0\}$ is selected. If $|A_{cs}| < k$, then all conflict sets are taken. Afterwards, the recomposing activity set $A_r = \cup(A_{cs,k}) \subseteq A_b(D)$ is created. We note that in the case where two conflict sets have the same occurrence frequency, a random one is chosen. This secondary criterion avoids bias, and gives better performances empirically than any other straightforward criteria.

Merge conflict graph (MCG) recomposes on conflicting activities that co-occur on the trace level by constructing a weighted undirected graph $G = (V, E)$ where $E = \{\{a_1, a_2\} \mid \exists_{\boldsymbol{\gamma} \in \Gamma_D}\ a_1 \in C(\boldsymbol{\gamma}) \wedge a_2 \in C(\boldsymbol{\gamma}) \wedge a_1 \neq a_2\}$ with a weight function $w : E \to \mathbb{N}^+$ such that $w((a_1, a_2)) = |\{\boldsymbol{\gamma} \in \Gamma_D \mid C(\boldsymbol{\gamma})(a_1) > 0 \wedge C(\boldsymbol{\gamma})(a_2) > 0\}|$ and $V = \{a \in A_b(D) \mid \exists_{(a_1,a_2)\in E}\ a = a_1 \vee a = a_2\}$. Then, with a threshold $t \in [0, 1]$, edges are filtered so that $E_f = \{e \in E \mid w(e) \geq t \times w_{\max}\}$ where $w_{\max}$ is the maximum edge weight in E. The corresponding vertex set and filtered graph can be created as $V_f = \{a \in A_b(D) \mid \exists_{(a_1,a_2)\in E_f} a = a_1 \vee a = a_2\}$ and $G_f = (V_f, E_f)$. Finally, the current net decomposition is recomposed on activity set $A_r = V_f$.

Balanced. This recomposition strategy extends the MFCS-k strategy but also tries to minimize the average size of the sub-nets resulting from the recomposition. For a border activity $a \in A_b(D)$, $|(a, D)| = |\cup_{S_i \in S_b(a,D)} A_v(S_i)|$ approximates the size of the recomposed sub-net on activity a. The average size of the recomposed sub-nets for a particular conflict set can then be approximated by $|(A_c, D)| = \frac{\sum_{a \in A_c} |(a,D)|}{|A_c|}$. The score of the conflict set can be computed as a weighted combination $\beta(A_c, D) = w_0 \times \frac{m(A_c)}{\max_{A'_c \in A_{cs}} m(A'_c)} + w_1 \times (1 - \frac{|(A_c,D)|}{\max_{A'_c \in A_{cs}} |(A'_c,D)|})$ where higher scores are assigned to frequent conflict sets that do not recompose to create large sub-nets. The activities of the conflict sets with the highest score, $A_r = \{a \in A_c \mid A_c \in \arg\max_{A'_c \in A_{cs}} \beta(A'_c, D)\}$, are then recomposed upon to create a net decomposition.

5.6.2. Log recomposition strategy

Similar to the net recomposition strategies, the existing IC strategy can be too lenient in including log traces which have conflicting activities that are unlikely to be resolved in the following decomposed replay iteration.

Strict include by conflict (SIC) increases the requirement for a to-be-aligned log trace to be selected for the next iteration. This addresses the limitation of IC which can include log traces whose merge conflicts are only partially covered by the net recomposition. Given the recomposed activity set A_r, SIC includes log traces as $L_r = [\sigma_i \in L \mid \forall_{a \in C(\gamma_i)} a \in A_r]$ with merge conflict if the corresponding conflict set is a subset of set A_r. However, this log strategy only works in conjunction with the net strategies that are based on conflict sets, i.e., MFCS-k and Balanced, so that at least one to-be-aligned log trace is included.

5.7. Experiment setup

Both synthetic and real-life datasets are used to evaluate the proposed recomposition strategies. Dataset generation is performed using the PTandLogGenerator [30] and information from the empirical study [32]; it is reproducible as a RapidProM workflow [68]. The BPIC 2018 dataset is used [74] as the real-life dataset. Moreover, two baseline net recomposition strategies are used: **All** recomposes on all conflicting activities, and **Random** recomposes on a random number of conflicting activities. Similarly, a baseline log recomposition **All** which includes all to-be-aligned log traces is used. For the sake of space, the full experimental setup and datasets are available at the GitHub repository[2] so that the experimental results can be reproduced.

5.8. Results

The results shed light on two key insights: First, the selection of the recomposition approach may lead to very different performances. Second, good performance requires both selecting appropriate conflicting activities and well-grouped to-be-aligned log traces.

Figure 5.5 presents the experimental results for both synthetic and real-life datasets. For each of the synthetic models, there are three event logs of different noise profiles described as net*X*-*noise probability*-*dispersion over trace* where $X \in \{1, 2, 3\}$. For the sake of readability, we only show the results of three out of five synthetic datasets, but the results are consistent across all five synthetic datasets). Readers interested in more details are referred to the GitHub link for a detailed explanation on noise generation and the rest of the experimental results. For the MFCS-k and Balanced strategies, only configurations using the SIC log strategy are shown; results showed that the SIC log strategy provides a better performance. For the others where SIC is not applicable, only configurations using the IC log strategy are shown as results indicated better performances. Overall, the

[2]See `https://github.com/wailamjonathanlee/Characterizing-recomposing-replay`

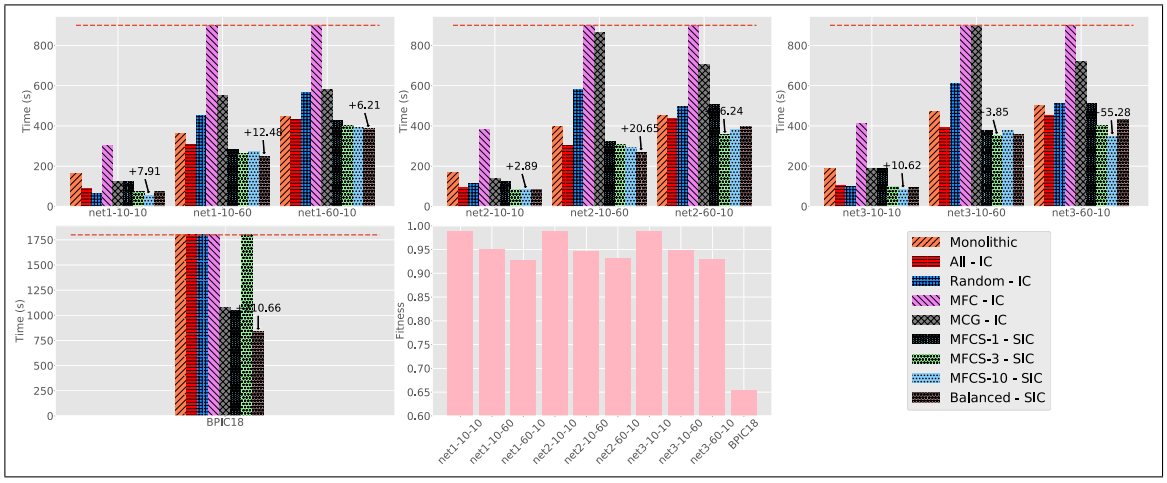


Figure 5.5. Bar chart showing fitness and overall time per net recomposition strategy (including the monolithic approach). The time limit is shown as a dashed red line and indicates infeasible replays. Best performing approaches and their time gains from the second fastest times are specified by black arrows.

results show that for both the monolithic and recomposition approach, it is more difficult to compute alignment results for less fitting datasets.

Different approaches give different performances. Comparing the monolithic and recomposition approach, it is clear that the recomposition approach provides a better performance than the monolithic counterpart under at least one recomposition strategy configuration. Furthermore, performance can vary significantly across different recomposition approaches. For example, the existing MFC strategy is the worst performing strategy where it is not able to give exact results for the real-life dataset and both the net$X$-10-60 and net$X$-60-10 noise scenarios of the synthetic datasets. The MFCS-k and Balanced strategies are shown to be the best performing strategies. While for high fitness scenarios, i.e., net$X$-10-10, MFCS-k give better performances with a high $k = 10$. This is because when there is little noise, it becomes simply a "race" to aligning traces with similar merge conflicts. Conversely, for low fitness scenarios, because merge conflicts are potentially much more difficult to resolve, the Balanced strategy avoids quickly creating large subcomponents that take longer to replay. In these cases, the time differences between the

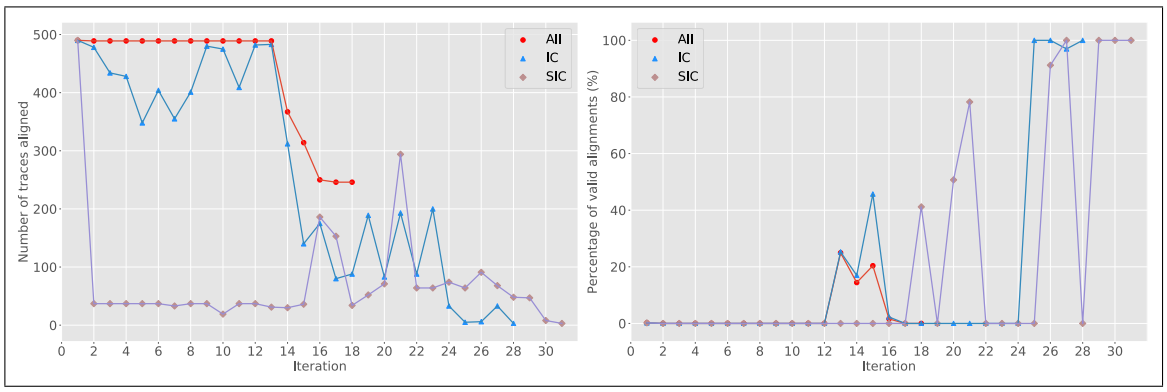


Figure 5.6. Comparing log strategies by showcasing the number of aligned traces (left) and percentage of valid alignments (right) per iteration on the real-life dataset BPIC18.

different feasible strategies can go up to three minutes. For all the experiments, the proposed recomposition strategies outperform the baseline strategies. Lastly, for the real-life dataset BPIC18, only the MFCS-1, Balanced, and MCG recomposition strategies are able to compute exact alignment results and the Balanced strategy outperforms MFCS-1 by more than three minutes.

Both net and log recomposition strategies matter. Figure 5.6 presents the number of aligned traces and percentage of valid alignments per iterations under All, IC, and SIC log strategies with net strategy fixed as Balanced on BPIC18. We first note that only the SIC log strategy resulted with exact alignment results. While all strategies start with aligning all traces in the first iteration, there are significant differences in the number of aligned traces across iterations. Similar to the All strategy, the existing IC strategy includes a high number of traces to align throughout all iterations; the number of aligned traces only tapered off in the later iterations as half of the traces have resulted as valid alignments. This confirms the hypothesis that the existing IC strategy can be too lenient with the inclusion of traces to align. Furthermore, up until iteration 13, none of the aligned traces reaches the necessary merging condition to result as a valid alignment; this means that both the All and IC strategies are "wasting" resources aligning many traces. Conversely, the SIC strategy keeps the number of traces to align per iteration comparatively lower. Moreover,

at the peak of the number of traces to align at iteration 21, almost 80% of the $\sim$300 aligned traces resulted as valid alignments. These are likely to explain why only the SIC log strategy is able to compute an exact result.

5.9. Related work

Performance problems related to alignment-based conformance checking form a well-known problem. A large number of conformance checking techniques have been proposed to tackle this issue. Approximate alignments have been proposed to reduce the problem complexity by abstracting sequential information from segments of log traces [61]. The notion of indication relations has been used to reduce the model and log prior to conformance checking [62]. Several approaches have been proposed along the research line of decomposition techniques. This include different decomposition strategies, e.g., maximal [65], and SESE-based [50]. Moreover, different decomposed replay approaches such as the hide-and-reduce replay [85] and the recomposition approach [38] have also been investigated.

Other than the alignment-based approach, there are also other conformance checking approaches. This includes the classical token replay [58], behavioral profile approaches [92] and more recently approaches based on event structures [26].

5.10. Conclusions

This chapter investigated the recomposition aspect of the recomposing conformance checking approach which can become a bottleneck to the overall performance. By defining the recomposition problem, we identify limitations of the current recomposition strategy in not fully resolving merge conflicts on the trace level and also being too lenient in the inclusion of log traces for the subsequent decomposed replay iteration. Based on the observations, three net recomposition strategies and one log recomposition strategy have been presented. The strategies were then evaluated on both synthetic and real-life datasets with

two baseline approaches. The results show that different recomposition strategies can significantly impact the overall performance in computing alignments. Moreover, they show that the presented approaches provide a better performance than baseline approaches, and both the existing recomposition and monolithic approaches. While simpler strategies tend to provide a better performance for synthetic datasets, a more sophisticated strategy can perform better for a real-life dataset. However, the results show that both the selection of activities to recompose on and log traces to include are important to achieve superior performances.

The results have shown that the recomposition strategy has a significant impact on performance. For the current and presented approaches, new net decompositions are created by recomposing the initial decomposition on selected activities. Entirely different net decompositions can be created using the merge conflict information from the previous iteration; however, our preliminary results showed that this may be difficult. Moreover, the advantage of merging on border activities rather than creating entirely different net decompositions is that the former guarantees in a number of finite steps, the framework will end up at the monolithic case which guarantees the merging condition.

Part III

Algorithm selection

6. Use of decomposition as a classification problem

6.1. Introduction

There can be many conformance checking algorithms for the same task. For example, for alignment-based techniques, there are many techniques that focus on being great at specific aspects, e.g., computation time [75], specific data type [44] and conformance issues [9, 6]. This means that the end user needs to be aware of their advantages and disadvantages to choose the appropriate technique for their data and objective. However, the end user may not have the expert knowledge to always select the most appropriate algorithm. One solution is to have an oracle that have been configured to help users with the decision given their objectives. This chapter presents our work in applying machine learning techniques to learn a classifier that would predict the best alignment algorithm in terms of whether to use decomposition or not to minimize computation time.

We tackle the algorithm selection problem by encoding it as a classification task. By applying well known classifiers, the trained predictive model can select the algorithm that is most likely to achieve best performance among the set of available algorithms. As a first instance, we tackle the problem of deciding when to apply decomposition-based algorithms to compute exact optimal alignments using A^* techniques so that computation time is minimized. The chapter presents the analysis of the trained models and identifies features that can have a significant impact on the performance of different algorithms. While this chapter tackles a specific problem in the space of alignment computation, it is easy to see that the proposed framework can have an impact on other scopes, i.e., tackle more general alignment algorithm selection problems, as well as in the considered algorithms to include other alignment-based algorithms such as planner-based approaches [19], and approaches based on different theories [1, 53].

The rest of the chapter is structured as follows: Section 6.2 presents the background and the general problem statement. Then Section 6.3 presents the specific problem of predicting when to apply decomposition-based algorithms as a classification problem to

be the focus of this chapter. Next, in Section 6.4, the experimental setup is presented. The results and analysis are then presented in Section 6.5. Section 6.6 presents the limitations. Finally, the chapter ends with the related work and conclusion in Section 6.7 and Section 6.8 respectively.

6.2. Background and general problem statement

The core idea of alignment-based techniques is to explain each executed event in terms of model activities so that the alignment is a sequence of event-activity pairs where each event is mapped to a step in the model [4, 17].

Definition 6.1 (The alignment problem)**.** *Let $A \subseteq \mathcal{U}_A$ be a set of observable process activities and $A_M = A \cup \{\tau\}$ is the set of model activities where $\tau \notin A$ denotes unobservable model activity. Then, given*

- *a trace of executed events $\sigma \in A^*$,*
- *a process model that contains the modeled behavior $M \subseteq {A_M}^*$, and*
- *a cost function $c : ((A \cup \{\gg\}) \times (A_M \cup \{\gg\})) \to \mathbb{Q}$ for alignment moves,*

the alignment problem is to use an algorithm $O \in \mathcal{O}$ to compute a valid alignment $O(\sigma, M) = \gamma \in (((A \cup \{\gg\}) \times (A_M \cup \{\gg\})) \setminus \{(\gg, \gg)\})^*$ such that the projection on the first element (ignoring any $\gg$) yields σ and the projection on the second element (ignoring any $\gg$) yields $\sigma_M \in M$. Moreover, alignment γ has minimal total cost, computed as $\sum_{(a,a_M)\in\gamma} c(a, a_M)$, amongst all valid alignments.

However, in the current literature, there are many available alignment algorithms that can resolve the alignment problem [4, 19, 9, 75, 6, 44, 38]. Given the potentially time consuming nature of the problem, a natural solution is to select the algorithm that gives optimal performance for the trace and model at hand.

Definition 6.2 (Per-instance alignment algorithm selection problem [8])**.** *Given a performance measure $m : ((A^* \times \mathcal{P}({A_M}^*)) \times \mathcal{O}) \to \mathbb{R}$, the per-instance alignment algorithm*

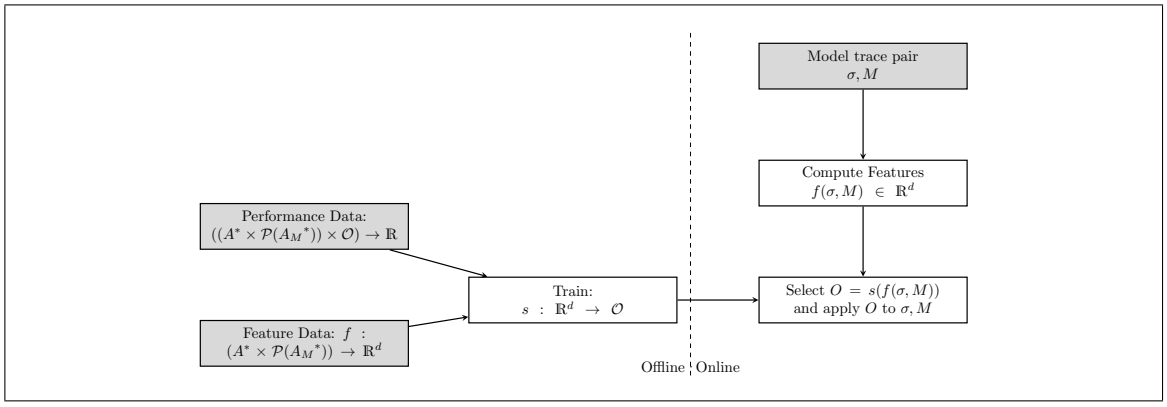


Figure 6.1. Workflow of alignment algorithm selection (adapted from [8])

selection problem is to find a mapping $s : (A^* \times \mathcal{P}(A_M{}^*)) \to \mathcal{O}$ *that optimizes the performance measure* $m(\sigma, M, s(\sigma, M))$ *by using the selected algorithm* $s(\sigma, M)$.

One way to tackle the problem is to learn such mapping through data generated from real execution of the alignment problem under different algorithms.

6.2.1. Using machine learning to learn algorithm selectors

Figure 6.1 presents the workflow taken by this chapter. The workflow is separated into an offline and online phase. At the offline phase, performance data is generated by running different alignment algorithms $O \in \mathcal{O}$ on model trace pairs σ, M. For each model trace pair, a d-dimensional feature vector can be generated using the function f. Using the performance and feature data, different machine learning techniques can be applied to learn the alignment algorithm mapping as a predictive model s. At the online phase, the predictive model is given the feature vector of a model trace pair and it should choose the best performing alignment algorithm to be applied.

6.3. Predicting the use of decomposition by classification

As previously introduced, this chapter focuses on the problem of deciding when to apply decomposition-based algorithms to compute exact optimal alignments using A* based

techniques so that computation time is minimized. This is achieved by applying the workflow presented in Figure 6.1 on a portfolio of four relevant alignment algorithms and using classifiers as the algorithm mapping.

6.3.1. Description of alignment algorithms

Classic A* algorithm (CLASSIC)**.** As presented in the seminal work [4], an optimal alignment between a log trace and petri net model is computed as a cost minimal complete firing sequence of their synchronous net product. This corresponds to a shortest path problem in the reachability graph of the net product and can be computed using the A* algorithm. In the classic approach, the marking equation is used as an admissible and consistent heuristic to approximate the remaining "distance" from a marking to the final marking. Further details can be found in the conformance checking book [17].

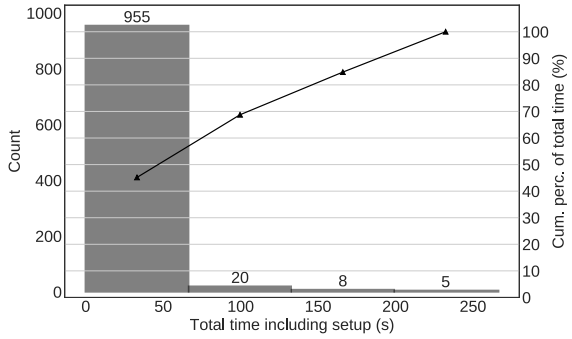
Classic A* algorithm with splitpoint heuristics (CLASSIC-SP)**.** This approach substitutes the marking equation with an extended marking equation [75]. It addresses a major limitation of the marking equation in having spurious markings as possible solutions by explicitly requiring valid markings (encoded as constraints) at pre-determined splitpoints. However, a limitation of the extended marking equation is the increased complexity from the additional constraints.

Recomposing conformance checking framework (RECOMPOSE)**.** Decomposition techniques break down a large and complex alignment problem into smaller sub-problems so that they can be solved simultaneously to obtain performance gains. In an iterative manner, the recomposing conformance checking framework decomposes the alignment computation between a log trace and model before merging the sub-alignment results [38]. For the decomposed alignment problems, the CLASSIC algorithm is used.

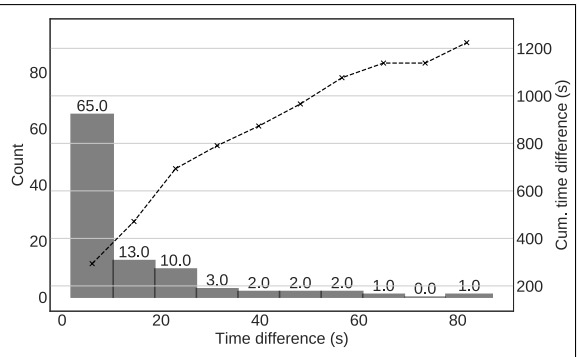
Recomposing conformance checking framework with splitpoint heuristics (RECOMPOSE-SP)**.** Since CLASSIC is used to solved decomposed alignment problems, the extended marking equation heuristics can also be applied.

6.3.2. Performance of the algorithms

There is potential in deciding the alignment algorithm on a model trace pair basis if there does not exist a single algorithm which dominates over all other algorithms for all model trace pairs. We show this by the experimental results of a synthetic dataset which includes the log *net1-60-60* with 1000 traces and model *net1* and have been presented in [36]. Alignment computation were performed using CLASSIC-SP and RECOMPOSE-SP [1]. Overall, accounting only valid alignments, CLASSIC-SP took 7726.23 seconds (s) to yield 988 valid alignments and RECOMPOSE-SP took 7753.64s to yield 993 valid alignments. This means that CLASSIC-SP resulted in 22 invalid alignments that were terminated due to exceeding a time limit of 300s per trace. This contrasts the 7 invalid alignments under RECOMPOSE-SP that were also terminated for the same reason.



(a) Alignment times and line graph of the cumulative percentage of total time per time bin of valid alignments



(b) Alignments with top 10% absolute time differences (computed as CLASSIC-SP - RECOMPOSE-SP) and line graph of the cumulative time difference

Figure 6.2. Histograms of alignment time statistics on model *net1* and log *net1-60-60*

However, focusing on the valid alignments computed by both algorithms, CLASSIC-SP took lesser time for 554 alignments in comparison to RECOMPOSE-SP. This suggests

[1]Experiments were performed single threadly on Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz with 16GB of allocated memory.

that CLASSIC-SP is faster than RECOMPOSE-SP in the general case. Yet, analyzing alignments with large absolute time differences gives a different picture. Figure 6.2b shows that for the 99 alignments with a top 10% performance difference, RECOMPOSE-SP outperformed CLASSIC-SP. Moreover, it illustrates that if RECOMPOSE-SP were used for these 99 alignments during the computation under CLASSIC-SP, $\sim 1200s$ would have been saved. This puts forth a strong case for being able to select the appropriate alignment algorithm depending on a given trace and model.

Given the feature and performance data captured from alignment computations under the four algorithms, a classification task is defined so that the trained classifier would select the algorithm that minimizes computation time.

Definition 6.3 (Best performance classification)**.** *Let* $(F_1, O_1), (F_2, O_2), \ldots, (F_n, O_n)$ *be data such that* $F_i = (F_i^1, F_i^2, \ldots, F_i^d) \in \mathcal{F} \subseteq \mathbb{R}^d$ *is a* d*-dimensional feature vector extracted from model* M_i *and trace* σ_i*.* $O_i \in \mathcal{O}$ *is the algorithm that resolves the alignment problem in the least amount of wall-clock time, i.e.,* $O = \text{argmin}_{O' \in \mathcal{O}}\, m(\sigma_i, M_i, O')$*. The best performance classification problem is to learn a function* $h : \mathcal{F} \to \mathcal{O}$*.*

Typically, a loss function L is used to evaluate the quality of a classifier, e.g., the empirical error rate $L_n(h) = \frac{1}{n} \sum_{i=1}^{n} I(h(F_i) \neq O_i)$. Referring to Definition 6.2, clearly a classifier that resolves the best performance classification problem will be a good estimator of the mapping that optimizes the performance measure of wall-clock time for the per-instance alignment algorithm selection problem.

6.3.3. Model features

As shown in Table 6.1, there are three categories of features which are extracted from the log trace, the model, and the synchronous net product between the trace net and model. To ensure that the predictive models can practically perform predictions prior to alignment, the features are relatively simple and can all be extracted in $\mathcal{O}(|V| + |E|)$ where $|V|$ and

$|E|$ are the sizes of the vertex and edge set of the synchronous net product. There are 67 features in total.

Table 6.1. Extracted features

Name	Category	Description
trace_length	trace	Trace length
n_activity	trace	No. of activities
activity_repeat_mean	trace	Repetition per activity - mean
activity_repeat_std	trace	Repetition per activity - std
n_transition	model/snp	No. of transitions
n_place	model/snp	No. of places
n_arc	model/snp	No. of arcs
n_inv_transition	model/snp	No. of invisible tran.
n_dup_transition	model/snp	No. of duplicated tran.
n_uniq_transition	model/snp	No. of unique tran.
inv_tran_in_degree_mean	model/snp	In deg. of invisible tran. (mean)
inv_tran_in_degree_std	model/snp	In deg. of invisible tran. (std)
inv_tran_out_degree_mean	model/snp	Out deg. of invisible tran. (mean)
inv_tran_out_degree_std	model/snp	Out deg. of invisible tran. (std)
uniq_tran_in_degree_mean	model/snp	In deg. of unique tran. (mean)
uniq_tran_in_degree_std	model/snp	In deg. of unique tran. (std)
uniq_tran_out_degree_mean	model/snp	Out deg. of unique tran. (mean)
uniq_tran_out_degree_std	model/snp	Out deg. of unique tran. (std)
dup_tran_in_degree_mean	model/snp	In deg. of dupl. tran. (mean)
dup_tran_in_degree_std	model/snp	In deg. of dupl. tran. (std)
dup_tran_out_degree_mean	model/snp	Out deg. of dupl. tran. (mean)
dup_tran_out_degree_std	model/snp	Out deg. of dupl. tran. (std)
place_in_deg_mean	model/snp	In deg. of places (mean)
place_in_deg_std	model/snp	In deg. of places (std)
place_out_deg_mean	model/snp	Out deg. of places (mean)
place_out_deg_std	model/snp	Out deg. of places (std)
n_and_split	model/snp	No. of AND splits
n_xor_split	model/snp	No. of XOR splits
n_biconnected_component	model/snp	No. of biconnected components
n_subnet	model	No. of subnets
subnet_n_tran_mean	model	No. of tran. in subnets (mean)
subnet_n_tran_std	model	No. of tran. in subnets (std)
subnet_n_inv_tran_mean	model	No. of inv. tran. in subnets (mean)
subnet_n_inv_tran_std	model	No. of inv. tran. in subnets (std)
subnet_n_dup_tran_mean	model	No. of dup. tran. in subnets (mean)
subnet_n_dup_tran_std	model	No. of dup. tran. in subnets (std)
subnet_n_uniq_tran_mean	model	No. of uniq. tran. in subnets (mean)
subnet_n_uniq_tran_std	model	No. of uniq. tran. in subnets (std)
subnet_n_place_mean	model	No. of place in subnets (mean)
subnet_n_place_std	model	No. of place in subnets (std)
subnet_n_arc_mean	model	No. of arc in subnets (mean)
subnet_n_arc_std	model	No. of arc in subnets (std)

6.3.4. Classifiers

Decision tree and random forests are used as classifiers for the advantage of yielding more interpretable results.

Decision tree (DT) [12] Decision trees recursively partitions a dataset via decision rules

at internal nodes so that each leaf node contains data points of the same target variable. However, two main limitations of this classifier with consideration to the dataset are that decision trees are prone to overfitting and can create biased trees if the dataset is unbalanced. The overfitting limitation is addressed by limiting the tree depth and setting the minimum number of samples required at a leaf node. Moreover, the dataset is balanced by adjusting the class weights of the samples so that their weights are inversely proportional to class frequencies.

Random forest (RF) [11] This is an ensemble method that fits a number of decision tree classifiers on sub-samples of the dataset and uses the average to improve the predictive accuracy and to avoid overfitting.

6.4. Experimental setup

The alignment experiments were performed single threadly on Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz with 16GB of allocated memory. 300 seconds was set as the timeout threshold for each trace. For the sake of space, further detail on the experimental setup, algorithm parameter configurations, and instructions on replication can be found at the GitHub repository [2]. Given that there are many parameter configurations for the four alignment algorithms, we relied on the findings reported by the related papers [36, 75] and large scale parameter optimization experiments [79, 80]. Some of the more important parameter configurations include the use of LP for heuristic computations and the use of second-order queueing criterion for the priority queue of the A* search [80]. For the algorithms CLASSIC-SP and RECOMPOSE-SP, due to a lack of guideline on parameter configurations, we determined the respective configurations following the example code [75] and small scale experiments. For the decomposition-based algorithms, two different initial decompositions were used for most of the model trace pairs.

[2] See `https://github.com/jwllee/predicting-alignment-algorithm-performance-using-machine-learn tree/v0.1`

Table 6.2. Statistics on models and logs used to produce predictive model data

	n_transition	n_place	n_arc	trace_length
mean	216.3	180.7	478.3	72.5
std	43.5	29.2	85.6	37.6
min	127.0	120.0	306.0	1.0
25%	188.0	163.0	424.0	49.0
50%	239.0	180.5	502.0	62.0
75%	253.0	207.0	546.0	86.0
max	264.0	245.0	580.0	419.0

6.4.1. Data description

Two published synthetic datasets were used to generate the necessary performance data for the predictive tasks [36, 39]. After verifying, cleaning and formatting the generated data, the final dataset records performance data for over 800,000 alignment computations on ∼140,000 model trace pairs that have been generated from 20 models. As shown in Table 6.2, the dataset tends towards larger models and logs for which the divide-and-conquer alignment approaches RECOMPOSE and RECOMPOSE-SP can leverage decompositions to improve performance. In terms of the trace lengths, there is a wide range where the minimum trace length is of 1 event and the maximum trace length is of 419 events.

6.4.2. Classification data classes

To establish the classes for the BEST-ALGO classification task, the wall-clock time (total time including setup) is used to identify the best performing algorithm for a given model trace pair with a prerequisite of the alignment being valid and cost minimal. Figure 6.3 shows the data class distribution of the dataset as the black bars. They show that for ∼70% of the model trace pairs, the CLASSIC computes in lesser time than the other three. We also considered that some traces might be duplicated across different event logs generated from the same model and created another dataset where duplicates were removed. Two model trace pairs are considered the same if they have the same extracted features and the same alignment costs (alignment length is not considered since there can

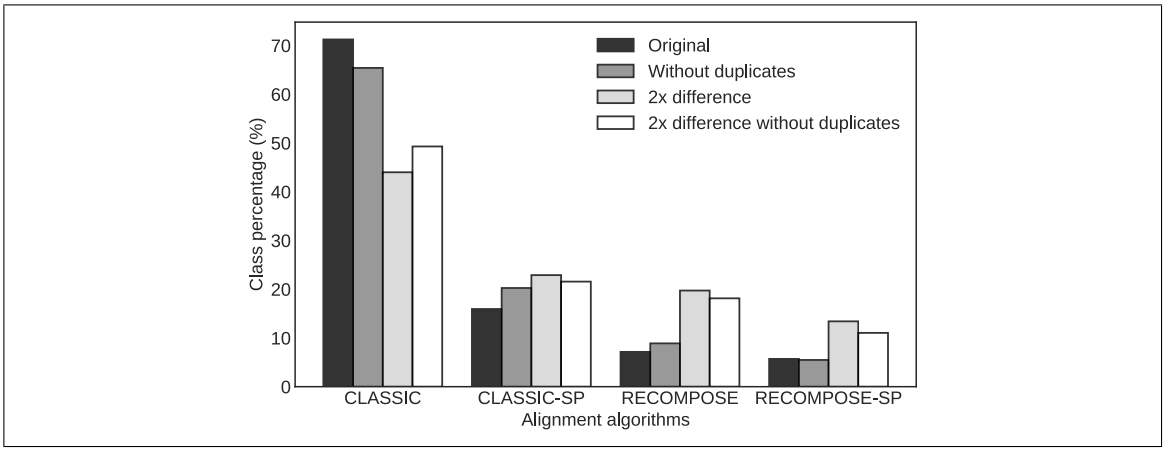


Figure 6.3. Class distribution of datasets

be multiple distinct cost minimal alignments). This resulted in a removal of 59,630 model trace pairs. The gray bars shows the class distribution of the *duplicate-free* dataset and that there are little differences with the original dataset in terms of class distribution.

Yet, not all data classes are equally important. By classifying the data points, the information on the time differences between algorithms is lost. Table 6.3 presents the time differences between algorithms after separation by data classes. Each row records the time differences statistics with respect to a particular data class on alignments that were valid across all four algorithms. Analyzing the time statistics, it is clear that some data classes and data class pairs are more interesting than others. For example, the mean time differences are very low for the instances where CLASSIC is the best performing algorithm. In contrary, for the instances where RECOMPOSE-SP is the best performing algorithm, the time differences are much larger and is at $\sim$16s with respect to CLASSIC and RECOMPOSE.

As Section 6.5 will further present, we found that the classifiers did not achieve a balanced performance across all data classes even after accounting for the class imbalance during training. One of the possible reasons is that there are instances where all four algorithms have similar performances. This means that while there is still an algorithm that finishes in the shortest time, including the data for these instances can actually add undesirable noise. To tackle this problem, a third dataset is created by only including instances

Table 6.3. Time difference statistics with respect to the best performing algorithm per data class

Data class	Count	Total Time including setup (s)															
		CLASSIC				CLASSIC-SP				RECOMPOSE				RECOMPOSE-SP			
		mean	std	se	infeasible	mean	std	se	infeasible	mean	std	se	infeasible	mean	std	se	infeasible
CLASSIC	174858	-	-	-	-	0.29	5.03	0.01	24	0.04	0.51	0.00	0	0.31	5.12	0.01	0
CLASSIC-SP	39118	9.67	19.03	0.10	22	-	-	-	-	8.25	16.76	0.08	0	0.75	5.35	0.03	0
RECOMPOSE	13979	0.84	8.60	0.07	32	2.87	15.46	0.13	16	-	-	-	-	3.36	18.18	0.15	0
RECOMPOSE-SP	17523	16.34	22.84	0.17	20	1.33	5.13	0.04	12	15.81	22.98	0.17	0	-	-	-	-

where there is at least a kx time difference between a pair of algorithms. Setting $k = 2$ left the original and duplicate free datasets with 86,469 and 74,582 instances respectively. As shown in Figure 6.3, the filtering also improved the class imbalance problem where now $\sim$50% of the instances belong to the CLASSIC class.

Lastly, across all data classes there is at least one algorithm which was unable to yield valid alignments for some traces. It would be interesting to investigate the reasons for these infeasible replays.

6.4.3. Evaluation

The classifiers are evaluated in two different ways:

- the classification performance in terms of precision, recall and F1-score
- the penalized average runtime with a penalty factor of 10, i.e., a timeout counts as 10 times the timeout (PAR10) [8]

The performance of each classifier is then compared to a stratified random classifier (Rand), the virtual best solver (VB), and the single best solver (SB). The stratified random classifier selects the algorithm according to the training set's class distribution. The virtual best solver assumes that we have a perfect classifier that can always choose the best algorithm for a model trace pair. The single best solver is the algorithm in the portfolio with the best overall performance in terms of PAR10. To ensure consistent evaluation, cross validation with the same 5 folds partition is used for all the experiments.

6.4.4. Model selection

For the decision tree classifier, model selection is performed using an exhaustive grid-search with cross validation of 5 folds to estimate the optimal split criterion (Gini impurity or entropy), tree depth (3 - 14) and minimum required samples for leaves (10 - 100, with steps of 10). For the random forest classifier, given that the bagging procedure finds the best set of decision trees, we tested a range of parameter values for the optimal number of estimators (10 - 1000, with steps of 10) and kept the default settings for the remaining parameters from the sklearn package [13]. For both classifiers, all 67 features presented in Table 6.1 are used.

6.5. Results

In this section, we present the experimental results and analysis. Overall, there were no major differences between the results from the original and duplicate-free dataset, and their 2x difference versions, therefore only the results on the original and 2x difference dataset are presented.

6.5.1. Classification performance

Table 6.4 presents the experimental results on the original dataset. Both decision and random forest achieved better precision than the baseline. However, decision tree has lower weighted average recall and F1-score than the baseline. This is due to the class imbalance where simply predicting CLASSIC frequently and neglecting the other classes can give better overall performance. In contrary, the decision tree classifier clearly tried to achieve good performances across all classes. Comparing decision tree and random forest, the more complex random forest outperformed decision tree by a large margin. Inspecting each class, we find that while random forest was able to achieve >0.50 precision for all classes, it has trouble recalling all the instances of each class. Figure 6.5a shows the normalized confusion matrix of the test set results on the original dataset under random

forest. It shows the recall problem for the CLASSIC-SP and RECOMPOSE-SP classes where the classifier erroneously classified CLASSIC-SP and RECOMPOSE-SP instances as CLASSIC. Through further inspection, the problem of instances having similar performances was identified and addressed using the kx difference filter.

Data class	Precision			Recall			F1-score			Support
	Rand	DT	RF	Rand	DT	RF	Rand	DT	RF	
CLASSIC	0.71	0.87	0.82	0.71	0.25	0.95	0.71	0.39	0.88	174858
CLASSIC-SP	0.16	0.21	0.64	0.16	0.52	0.32	0.16	0.30	0.43	39118
RECOMPOSE	0.06	0.50	0.59	0.06	0.68	0.65	0.06	0.58	0.62	13979
RECOMPOSE-SP	0.07	0.13	0.58	0.07	0.60	0.28	0.07	0.22	0.38	17523
Weighted avg.	0.54	0.69	0.76	0.54	0.34	0.79	0.54	0.37	0.76	245478

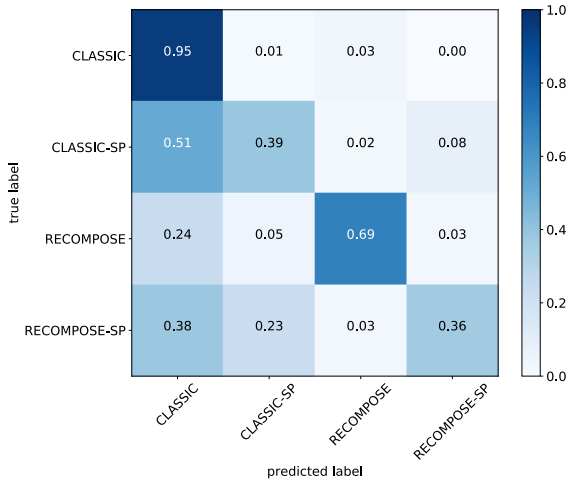
(a) Original dataset

Data class	Precision			Recall			F1-score			Support
	Rand	DT	RF	Rand	DT	RF	Rand	DT	RF	
CLASSIC	0.23	0.37	0.70	0.23	0.73	0.58	0.23	0.49	0.63	19789
CLASSIC-SP	0.44	0.65	0.66	0.44	0.28	0.81	0.44	0.39	0.73	38034
RECOMPOSE	0.14	0.65	0.71	0.13	0.64	0.66	0.13	0.65	0.69	11592
RECOMPOSE-SP	0.20	0.42	0.57	0.20	0.48	0.43	0.20	0.45	0.49	17054
Weighted avg.	0.31	0.54	0.66	0.31	0.47	0.66	0.31	0.46	0.65	86469

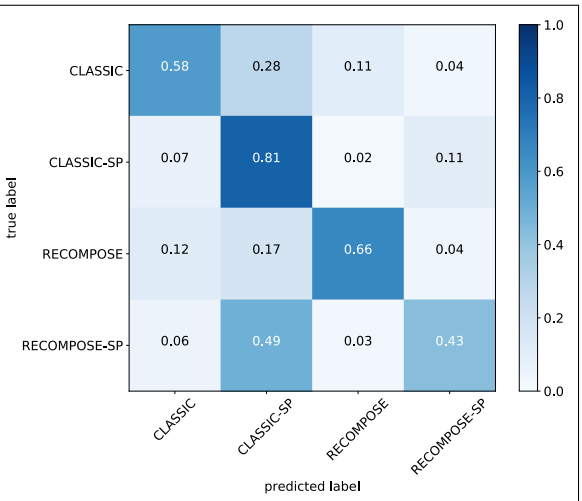
(b) 2x difference dataset

Figure 6.4. Experimental results of Rand, DT, and RF on two datasets

Table 6.4b presents the experimental results on the 2x difference dataset. This time both decision and random forest achieved better or equal performances with respect to the baseline across all weighted average metrics. With the exception of the recall and F1-score for the CLASSIC-SP class, the decision tree classifier achieved better performance across all metrics and classes.



(a) Original dataset



(b) 2x difference dataset

Figure 6.5. Normalized confusion matrices under random forest on two datasets

Different to the unfiltered dataset results, the performance differences between decision tree and random forest are much smaller. In fact, the weighted average performance of the random forest classifier went down more than 0.1 for all three metrics. However, inspecting the normalized confusion matrix shown in Figure 6.5b show that the recall problem with the CLASSIC class has been largely resolved. Yet, the classifier still erroneously classified a large number of RECOMPOSE-SP instances as CLASSIC-SP. A likely explanation is that, similar to before, these erroneously classified instances yield similar performances under CLASSIC-SP and RECOMPOSE-SP but different performances under CLASSIC and RECOMPOSE. However, clearly the random forest is able to distinguish between model trace pairs that should/shouldn't use the splitpoint heuristics and decomposition. In the following, we present the discriminating features that the classifiers used to achieve the reported performances.

6.5.2. Algorithm performance

Table 6.6a presents the average PAR10 scores in seconds per solver. None of the classifiers was able to beat the single best (SB) solver which was the RECOMPOSE-SP algorithm for both datasets. For the original dataset, both classifiers had lower average PAR10 scores than the random classifier, with the decision tree classifier being the closest to the SB solver (0.3s difference). For the 2x difference dataset, the random forest classifier was able to achieve comparable performance to the SB solver and is almost twice as fast as the random and decision tree classifiers. The decision tree classifier achieved worse performance than the SB solver. Table 6.6b presents the number of feasible replays per solver. The results are similar to the average PAR10 scores.

6.5.3. Analysis of feature importance

Analyzing the feature importance learned by the classifiers can provide insights into the discriminating factors that decide the best algorithm selection. Feature importance is

(a) Average PAR10 scores (s)

	Original	2x difference
VB	1.04	2.24
SB	1.57	3.57
Rand	3.89	6.67
DT	1.87	6.34
RF	2.22	3.83

(b) Number of feasible replays

	Original	2x difference
VB	245,478	86,469
SB	245,478	86,469
Rand	245,424	86,437
DT	245,476	86,437
RF	245,470	86,462

Figure 6.6. Algorithm performance in terms of PAR10 scores and the number of feasible replays. The single best solver (SB) is the RECOMPOSE-SP for both datasets.

computed as the expected fraction of samples that a feature influences in their final prediction decision, i.e., a feature that is used as a decision node near the top of the tree will have greater feature importance relative to one that is used lower down. Figure 6.7 presents the top five important features of the trained decision tree and random forest for the 2x difference dataset. While they are different for both classifiers, two and four out of the five features are related to the number ingoing and outgoing arcs of places. The ingoing and outgoing arc statistics of places indicate the amount of choices, e.g., a place with two outgoing arcs is an XOR split to two choices. This suggests that the amount of XOR decision points is a major deciding factor. At first glance, this is a surprising result because a large amount of parallelism will greatly increase the search space for a cost minimal alignment and it should be much easier to identify a large amount of parallelism through the ingoing and outgoing arc statistics of transitions rather than places. However, the expanded search space from parallelism generally does not harm the performance of A* approaches since the actual solution vector can typically be concluded directly from the solutions to the LP problems. In contrary, choices would lead to additional LP computations in the search space.

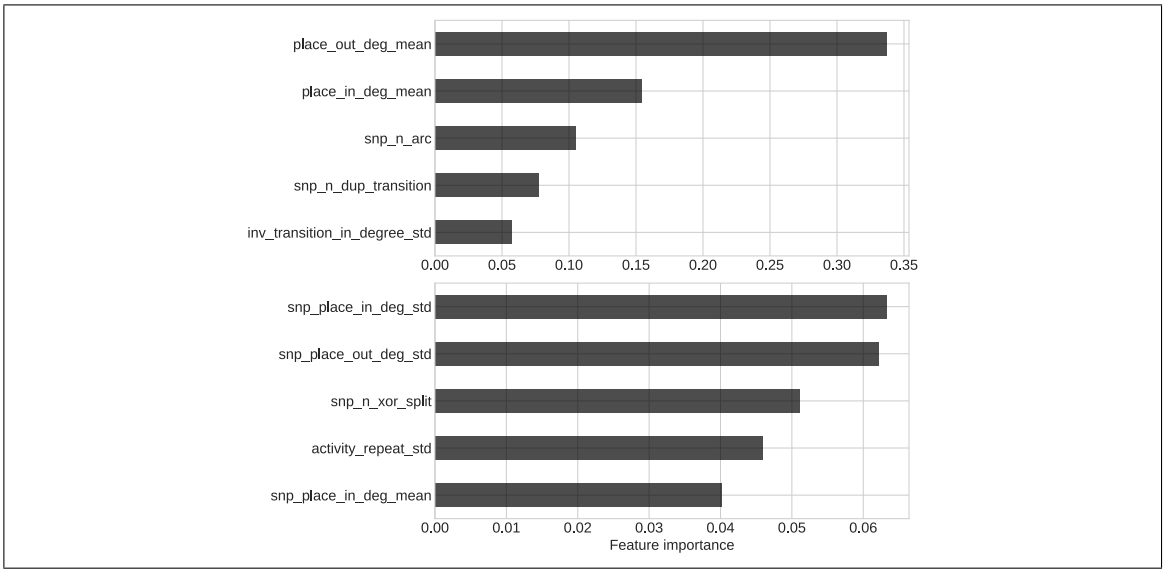


Figure 6.7. Top five features of decision tree (top) and random forest (bottom) trained on 2-difference dataset

6.5.4. Analysis of infeasible instances

Lastly, model trace pairs whose alignment were infeasible under one or more algorithms are analyzed. In the original dataset, CLASSIC and CLASSIC-SP had 74 and 52 infeasible instances under a 300s threshold respectively while RECOMPOSE and RECOMPOSE-SP had none.

Using the observation presented in [75] and the noise generation information of the datasets, we were able to verify for multiple instances that there was a recurrent problem of swapped events leading to CLASSIC performing badly. Furthermore, for these instances, both CLASSIC-SP and RECOMPOSE-SP were able to compute the alignments quickly due to the enforcing of valid markings as a constraint in the linear program (LP) [75]. At times, RECOMPOSE was also able to achieve similar performance. A likely reason is that the decomposition was able to capture the swap deviation within a relatively small subcomponent.

For CLASSIC-SP, the timeout was clearly due to the number of LPs that had to be solved.[3] This stems from expanding markings with no solution vector so that an incrementally larger LP has to be solved to yield an underestimate cost from the marking to the final marking [75]. For the 52 infeasible replays, the mean number of restarts was 46 with a minimum and maximum of 28 and 81. This contrasts feasible replays of non-perfectly fitting traces under CLASSIC-SP which has a mean of 11 and a minimum and maximum of 3 and 66.

6.6. Limitations

There are a number of limitations with the presented results. With respect to the proposed prediction problem, this chapter addressed a rather limited aspect in the alignment algorithm selection problem space. Moreover, the experimentation is limited by the characteristics of the model and log dataset used to generate the feature and performance data. There is also a practical constraint with respect to the available computational resources so that the dataset does not have valid alignments for traces that require more than 300 seconds to align.

With respect to the proposed predictive task, one limitation is in the use of total computation time as the sole performance indicator. While minimizing computation time is the main goal, it is not a consistent measure for training predictive models. A better choice would be more consistent measures such as the number of visited states. However, due to the potentially long time that heuristic computation may take, selecting a good performance indicator other than time is not easy. It might be interesting to use predictive models that give multi-outputs so that a vector containing all these measures is predicted. Another limitation is in terms of the selected features. Potentially more informative features could have been included, especially with respect to log traces. Moreover, while ensuring that feature computation is fast allows the predictive model to be practically useful at runtime, another approach is to allow the inclusion of more informative features that are more

[3] LPs are solved rather than ILPs due to parameter configuration

costly to compute but take into account their costs by including that into the learning of the predictive models [31].

Furthermore, while the filtering by kx time difference resulted in better results, a minor limitation is that the filtering does not enforce the kx difference to be between the best performing algorithm and another algorithm. Another possible solution could be creating new classes for model trace pairs that yield similar performances under different subsets of algorithms.

6.7. Related work

6.7.1. Existing approaches

There is a large number of different approaches that compute cost minimal alignments. Many are approaches which tackle the equivalent shortest path graph problem using A* based algorithms [4, 19, 75]. Non A* based approaches have also been proposed. It has been shown that symbolic algorithms can be used to compute alignments after a preprocessing step of the model [9]. Event structure based approaches have also been proposed [6]. For acyclic process models, the alignment problem can be encoded as a Constraint Satisfaction Problem (CSP) [44]. Decomposition approaches have also been shown to achieve performance gains over their monolithic counterpart [38].

6.7.2. Parameter tuning and algorithm selection

Many process mining algorithms can require in-depth knowledge to apply. Moreover, it can be difficult to set the optimal parameter configuration. Empirical studies have been performed to describe the effects and trends of different parameter configurations of the classic A* algorithm for different populations of process models [80]. The results have shown that there exist parameter configurations that have a significant positive impact on performance. A similar problem exists on the algorithm level for process discovery where it can be difficult to decide the appropriate discovery algorithm for a particular event log. A

recommender system that uses portfolio-based algorithm selection strategies was proposed to tackle the problem [55]. Another framework make use of reference models to select the process mining algorithm that would mine models that are likely to have high similarity to the provided models [90].

6.8. Conclusion

The time consuming nature of alignment computation has led to the proposal of several alignment algorithms that can have dramatically different performance under different scenarios. This means that it can be difficult to identify the algorithm that will terminate in minimal time.

This chapter laid out the general problem of algorithm selection for conformance checking and focused on a more specific problem of deciding when to apply decomposition-based algorithms to compute exact optimal alignments using A^* based techniques so that computation time is minimized. This is achieved by encoding the algorithm selection problem as a classification task of selecting the best performing algorithm per model trace pair basis. The proposed approach has been empirically validated using decision tree and random forest classifiers and with data generated from a large number of alignment computations using publicly available datasets.

Part IV

Online conformance checking

7. A HMM-based approach to online conformance checking (HMMConf)

7.1. Introduction

In this chapter we turn to the challenge of conformance checking under an online context. Given the volume and velocity at which event data comes in, organizations may not store these data for offline analysis and have to resort to online techniques. Moreover, performing analysis in real time allows process stakeholders to react to conformance issues.

Most existing conformance checking techniques require the trace of events to correspond to a completed case. This means that these techniques target offline scenarios and do not typically cater for online contexts where it is desirable to raise alerts as soon as a significant deviation is observed for cases that have not reached completion. Moreover, due to the continuous increase in recorded data, it can be infeasible for organizations to store data for offline processing. For example, Wal-Mart is estimated to collect more than 2.5 petabytes of data every hour from its customer transactions [27]. As such, in recent years, a new set of algorithms [78, 15, 16] has been proposed for *online* scenarios in which we assume to have an *event stream* as input so that each item relates to an observed event for a case. Here, we propose a novel online approach which performs conformance checking on an event stream with constraints on memory and time.

There are several works on online conformance checking [78, 15, 16], but there still exists areas for improvement. For example, prefix alignments [78] and a similar approach based on enriching a transition system using alignment concepts [15] have difficulties handling warm start scenarios. Another approach [16] that performs conformance checking on behavioral patterns can lose information due to its abstraction.

One fundamental challenge of explaining the conformance of a running case is in balancing between making sense at the process level as the case reaches completion and putting emphasis on the current information at the same time. This can be illustrated through a running example. Figure 7.1 and Figure 7.2 show a process model and some

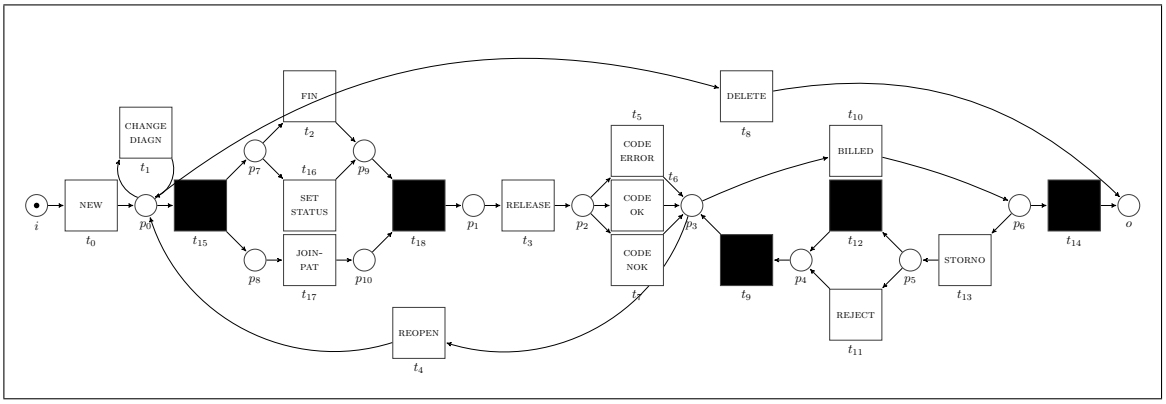


Figure 7.1. Running example: Petri net model

$\sigma_0 = \langle$NEW, FIN, SET STATUS, RELEASE, CODE OK, BILLED$\rangle$,
$\sigma_1 = \langle$NEW, FIN, SET STATUS, RELEASE, CODE OK, BILLED, STORNO, REJECT, BILLED$\rangle$,
$\sigma_2 = \langle$NEW, CHANGE DIAGN, FIN, SET STATUS, RELEASE, CODE NOK, CODE OK, BILLED$\rangle$,
$\sigma_3 = \langle$NEW, FIN, SET STATUS, RELEASE, CODE NOK, REOPEN$\rangle$,
$\sigma_4 = \langle$NEW, CHANGE DIAGN, FIN, RELEASE, CODE OK, MANUAL, RELEASE, CODE OK, REOPEN, DELETE$\rangle$,
$\sigma_5 = \langle$RELEASE, CODE OK, BILLED$\rangle$,
$\sigma_6 = \langle$NEW, FIN, RELEASE, CODE OK, BILLED$\rangle$,
$\sigma_7 = \langle$NEW, FIN, JOIN-PAT, SET STATUS, JOIN-PAT, RELEASE, CODE OK, BILLED$\rangle$

Figure 7.2. Running example: Traces

potential traces of the billing process of a hospital. This example is based on a real-life dataset [45] and a description of each activity in the process is presented in Table 7.1. As shown by the process model, ideally, each instance of the process corresponds to the billing process of a particular patient. As shown by trace σ_0 in Figure 7.2, after undergoing different medical services, these services are collected in a billing package and the package is released so that the patient can be billed. However, different scenarios can potentially occur during the process. For example, an invoice has to be sent to the insurance company of the patient and the invoice can potentially be rejected (REJECT) as shown by trace σ_1. Given the trace of a complete case, alignment-based techniques [75, 4, 6] excels at giving a globally optimal conformance solution. For example, for trace σ_5, cost-based alignment would show that it is missing the activities leading up to the RELEASE activity. However, as the case unfolds in an online scenario, alignment techniques can be slow to realise

Table 7.1. Activity description of the hospital billing event log taken from [46]

Activity	Description
NEW	A new billing package is created.
FIN	The billing package is closed, i.e., it may not be changed anymore.
RELEASE	The billing package is released to be sent to the insurance company.
CODE OK	A declaration code was successfully obtained.
BILLED	The billing package has been billed, i.e., the invoice is sent out.
CHANGE DIAGN	The diagnosis that the billing package is based on was changed.
DELETE	The billing package was deleted.
REOPEN	The billing package was reopened, i.e., additional medical services may be added or existing services removed.
CODE NOK	The declaration code was obtained with an error message.
STORNO	The billing package was canceled.
REJECT	The invoice sent to the insurance company was rejected.
SET STATUS	The status (i.e., new, closed, etc.) was manually changed.
EMPTY	The billing package is declared empty.
MANUAL	The billing package was manually changed from a non-standard system.
JOIN-PAT	The billing package was joined together since they refer to the same patient.
CODE ERROR	The declaration code could not be obtained.
CHANGE END	The projected end date of the billing package was changed.

such eventual explanation since they are always seeking a globally optimal explanation. Moreover, there is no flexibility in allowing warm start scenarios such as trace σ_5. At the other end of the spectrum, focusing only on the current information given by an incoming event of a case can be insufficient in providing a conformance explanation coherent at the process level. For example, trace σ_7 presents conformance issues with activities FIN, JOIN-PAT, and SET STATUS. However, if we only check directly following behavioral patterns as new events of the case come in, the conformance issue would not be detected since new events always form a modeled directly following behavioral pattern with the previous event. As such, it is desirable to have an online framework that yields balanced conformance explanations.

In this chapter, we present such framework based on Hidden Markov Models (HMM). As new events come in for running cases, the model alternates between localizing the running case within the reference model using the observed event and computing conformance from such estimated position. Different to the assumption of the standard HMM, both the previous state and observation can influence the next state due to non-conformance. This is modeled by conditioning state transition and observation probabilities by both the previous state and observation. Furthermore, rather than deciding beforehand the effects

of non-conformance, an Expectation-Maximization (EM) algorithm is applied to compute the parameters from past data.

The rest of the chapter is structured as follows: Section 7.2 presents the proposed technique. Section 7.3 details the parameter computation and estimation of the proposed technique. Section 7.4 presents the experimental evaluation of the proposed technique. Section 7.5 illustrates the application of the proposed technique on a real-life dataset. Section 7.6 presents the related work. Finally, Section 7.7 presents the conclusion.

7.2. Proposed technique

The proposed technique is based on representing the conformance checking scenario using a modified Hidden Markov Model (HMM). Given a process model and a stream of events, the approach represents process instances by a probabilistic estimation of their locations (state) within the process model and performs two tasks per event: orientation and conformance. This online procedure is supported by an offline component that is performed over past data as illustrated by the diagram in Figure 7.3. In this section, we present the online procedure which is initiated by an observable unit from the event stream, denoted by the grayed box, and then passed onto the two tasks of orientation and conformance, denoted by the subsequent three boxes.

Given the natural connection between Petri nets and HMMs, we use markings as possible states of a process instance. However, note that we can take further approximations by partitioning the set of possible markings into a smaller set of classes.

7.2.1. Overview

Algorithm 1 presents an overview of the online procedure. When a new observable unit comes in from the event stream (line 2), it first goes through the orientation phase. In this phase, either the previous state estimation or an initial state estimation of the case is retrieved depending on whether if the corresponding case has been previously observed

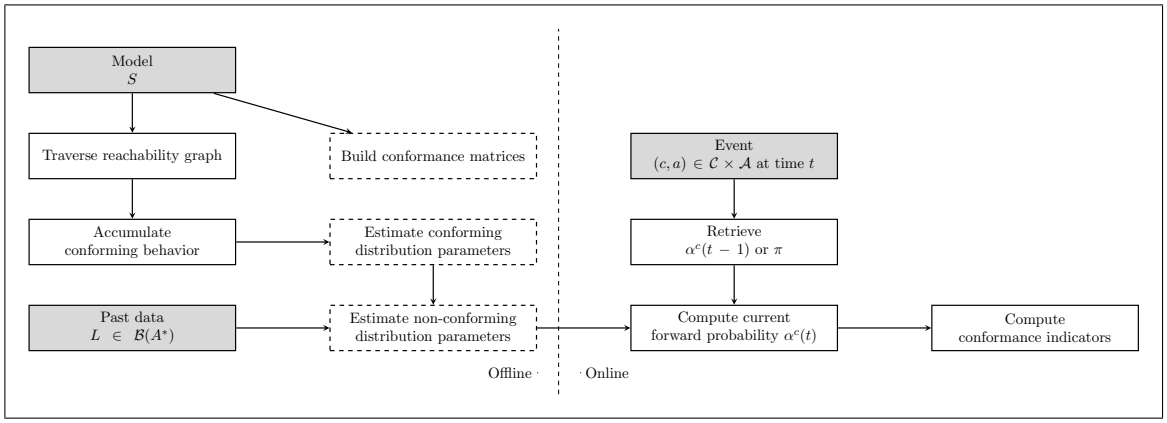


Figure 7.3. Overview of the proposed approach. The online component is presented in Section 7.2 and the offline component is presented in Section 7.3.

(line 3). Then, state estimation is performed, taking into account the new information given by the new observable unit (line 4). Afterwards, it moves onto the conformance phase where various conformance indicators are computed (line 5). Lines 2 – 5 correspond to the four boxes in the previous flow diagram in Figure 7.3.

Algorithm 1: Overview of online conformance computation

Input: S: stream of observable units

1 **forever do**

2 $(c, a) \leftarrow observe(S)$; `// New caseid-activity pair from the stream`

`// Phase 1: Orientation`

3 $state_c \leftarrow get\_either\_previous\_or\_initial\_state\_estimation(c)$;

4 $state_c \leftarrow update\_state\_estimation(state_c, a)$;

`// Phase 2: Compute online conformance values`

5 $compute\_conformance\_indicators(state_c, a)$;

7.2.2. Walk-through of an example

Here we walk through a case corresponding to trace σ_6 in Figure 7.2 to give some intuition. At the start of the case, one could assume that the case would be at the initial marking ($[i]$) so that its initial state estimation is a one-hot vector $\vec{1}_{[i]}$ with n components where n corresponds to the number of possible locations. Then, suppose we observe the first event (NEW). Clearly, this is described by the model and corresponds to firing

transition t_0. This yields a perfect conformance value. Similarly, the next event (FIN) puts the case at the state estimation of $\vec{1}_{[p_0]}$ (marking $[p_0]$ eventually enables activity FIN by firing invisible transitions) with perfect conformance value.

Suppose we observe the next event (RELEASE). Clearly, this is not conforming to the modeled behavior. To yield a plausible explanation for the non-conforming behavior, we perform two tasks to estimate the current state. First, given the previous event (FIN) and the assumption that it was perfectly conforming, the case must be currently at marking $[p_8, p_9]$. Second, checking the possible observations at marking $[p_8, p_9]$ would tell us that RELEASE does not correspond to an enabled transition. Incorporating both the state transition and the current event, we would estimate that the current state to be at $\vec{1}_{[p_8,p_9]}$ and that the event is completely non-conforming to the model. However, since the current observation is not described by the modeled behavior, we can no longer rely on the model to estimate the next state following the event (RELEASE).

Suppose we observe the next event (CODE OK). Similar to the previous event, we first estimate its current state given its previous event (RELEASE) and conformance. Since the previous conformance shows to be completely non-conforming to the modeled behavior, we cannot use the model as the basis for the state estimation. Instead, we assume to have a categorical distribution that estimates the next state given the previous event and state. This distribution is learnt from past data using a procedure that is presented later in the chapter. Suppose that in the past, there are similar cases which later turned out to have skipped the execution of activity JOIN-PAT. This would encourage us to estimate the current state to be at marking $[p_2]$. Second, we incorporate the current event (CODE OK) in our estimation. Since the transition corresponding to activity CODE OK is in fact enabled at marking $[p_2]$, observing CODE OK reinforces our estimation of the current state to be at marking $[p_2]$. In contrary, if the observed event corresponds to activity RELEASE, then we might reallocate some probability mass from marking $[p_2]$ to marking $[p_1]$ rather than reinforcing our estimation on marking $[p_2]$. In this case, reinforcing our state estimation

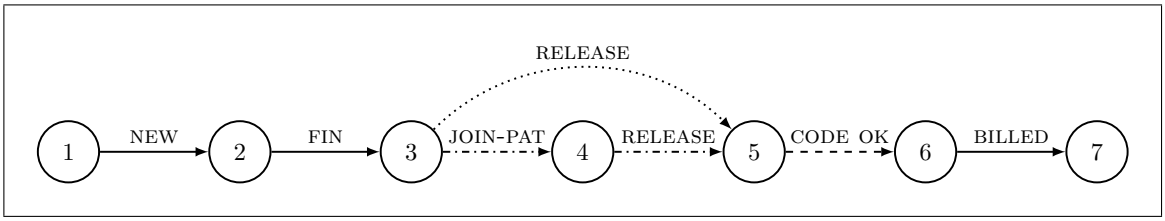


Figure 7.4. State estimation taken throughout trace σ_6 in Figure 7.2. Line style indicates the conformance explanation of the corresponding execution where a solid line indicates complete conformance, a dotted line indicates complete lack of conformance, a dashed line indicates moderate conformance, and a dash-dotted line indicates a possible model execution that non-conforming observation might be referring to.

on marking $[p_2]$ yields a high conformance on the current event (CODE OK). In fact, the following event BILLED will further confirm our estimation.

Figure 7.4 illustrates the state estimations taken throughout the case. The line style of the arrows indicates the conformance explanation of the corresponding execution. For example, since we assume that the initial state is $\vec{1}_{[i]}$, the first event (NEW) and the second event (FIN) is completely conforming with the modeled behavior. In contrary, the third event (RELEASE) is completely non-conforming since it skipped over the activity JOIN-PAT. As previously explained, the event RELEASE brings the state estimation to allocate a high probability on marking $[p_2]$ where the transition corresponding to activity CODE OK is enabled. This leads to a value in between complete conformance and non-conformance. The last event has a high conformance value since the corresponding transition is enabled at the previous state estimation.

7.2.3. HMM-based conformance checking

As previously explained, the current event and an estimation of the case's current state are required to compute the conformance of the current observation. We define a conformance function as a mapping from a tuple of the current state estimation and observation to a value between 0 and 1.

Definition 7.1 (Conformance function)**.** *Let Z denote the set of possible states. conf* : $\mathbb{R}^{|Z|} \times A \to [0, 1]$ *is a function that maps a state estimation and an activity execution to a value between 0 and 1 so that a value near 0 indicates complete non-conformance and a value near 1 indicates complete conformance.*

For the model in Figure 7.1, *conf*$(\vec{1}_{[i]}, \text{NEW}) = 1$[1] yields complete conformance on the one-hot vector with 1 at the initial marking since transition t_0 is enabled.

As recalled, we perform a state estimation each time we observe a new event. Moreover, under perfect conformance, the current state is dependent solely on the previous state. We extend this assumption to the scenario of non-perfect conformance to meet the computational constraints of online processing. This means that the chain of state estimations corresponds to a Markov chain. However, under the scenario of non-perfect conformance, one cannot directly observe a case's state (hence the need for *state estimations*). Instead, the case's state is hidden or so-called *latent* so that we have to infer it from the observed events. The discrete progression of cases and the dependence between states of different time steps yields a modified HMM.

Definition 7.2 (HMM for conformance checking (HMMConf))**.** *Let Z be a set of latent states and let X be a set of observations. Let $W_{x_k} \in \mathbb{R}^{|Z|\times|Z|}$ be a state-transition probability matrix given observation $x_k \in X$, $V \in \mathbb{R}^{|Z|\times|X|}$ be an emission probability matrix under conforming conditions and let $W^d_{x_k}$ and V^d be their counterparts under non-conforming conditions. Let conf* : $\mathbb{R}^{|Z|} \times \mathcal{A} \to [0, 1]$ *be a conformance function and let $\pi \in \mathbb{R}^{|Z|}$ be the initial state distribution.*

$(W_{x_1}, \ldots, W_{x_{|X|}}, W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V, V^d, \textit{conf}, \pi)$ *is a modified hidden Markov model with states dependent on previous observations [43] so that:*

$$P(x_t \,|\, z_t, \ldots, z_1, x_{t-1}, \ldots, x_1) = P(x_t \,|\, z_t), 1 \leq t \leq T \tag{7.1}$$

[1] $\vec{1}_{[i]}$ is a one-hot vector with n components where $n = 7$ is the number of states in the reachability graph of the Petri net in Figure 7.1. $[i]$ is a multiset that denotes the state with 1 in $\vec{1}_{[i]}$; all other states has 0.

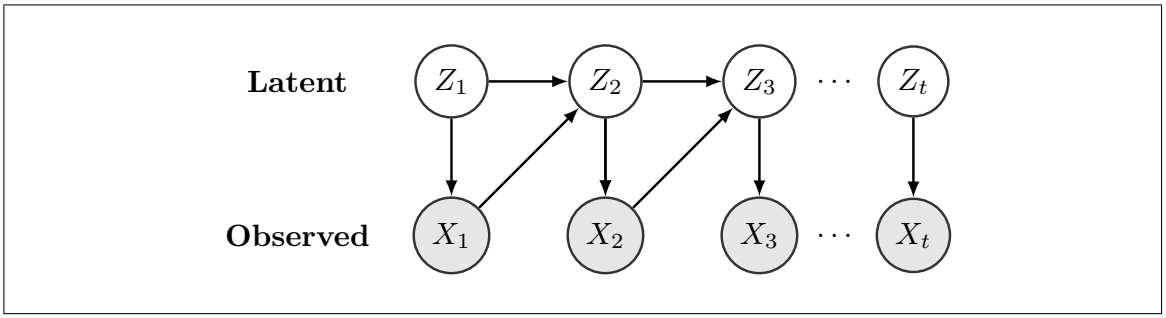


Figure 7.5. Graphical representation of HMMConf

$$P(z_t \,|\, z_{t-1}, \ldots, z_1, x_{t-1}, \ldots, x_1) = P(z_t \,|\, z_{t-1}, x_{t-1}), 2 \leq t \leq T \tag{7.2}$$

Figure 7.5 presents a graphical representation of HMMConf. In terms of dependencies between random variables, the difference between HMMConf and a standard HMM is the extra dependency between the current state Z_t and the previous observation X_{t-1}. As previously explained, the latent states are the possible markings in the Petri net model and the observations are the activities. We now present the state-transition probability (Eq. 7.2) and the observation probability (Eq. 7.1).

Definition 7.3 (Conformance dependent state-transition probability)**.** *Let W_{x_k} be the state-transition probability matrix given observation $x_k \in X$, $W^d_{x_k}$ be the probability matrix for the deviating behavior, and $\hat{\vec{z}}_{t-1}$ be an estimation of time $t-1$.*

$$\begin{aligned} w_{i,j}(k) &= P(Z_t = j \,|\, Z_{t-1} = i, X_{t-1} = x_k) \\ &\approx \textit{conf}(\hat{\vec{z}}_{t-1}, x_k) W_{x_k,i,j} + [1 - \textit{conf}(\hat{\vec{z}}_{t-1}, x_k)] W^d_{x_k,i,j} \end{aligned}$$

This is an approximation of the latent state conditional probability because W_{x_k} is a substochastic matrix; the final marking of a Petri net corresponds to an absorbing state that does not transition to other states once it is reached. To simplify the notations, we assume that the latent state and observation sets are ordered so that members can be referenced by their indices. Also, $z_{1:t} \equiv z_1, z_2, \ldots, z_{t-1}, z_t$. Component i of the estimation of the previous state $\hat{\vec{z}}_{t-1}$ is computed as $P(Z_{t-1} = i \mid X_{1:t-1} = x_{1:t-1}) = \frac{\alpha_i(t-1)}{\sum_k \alpha_k(t-1)}$ where $\alpha_i(t) = P(X_{1:t-1} = x_{1:t-1}, Z_{t-1} = i)$ is the forward probability.

Definition 7.4 (Conformance dependent observation probability)**.** *Let V be the observation probability matrix, V^d be the probability matrix for the deviating behavior, $x_k \in X$ be an observation, and $\hat{\vec{z}}_t$ be an estimation of the previous state.*

$$\begin{aligned} v_j(k) &= P(X_t = x_k \,|\, Z_t = j) \\ &\approx \textit{conf}(\hat{\vec{z}}_t, x_k) V_{j,k} + [1 - \textit{conf}(\hat{\vec{z}}_t, x_k)] V^d_{j,k} \end{aligned}$$

Similar to the state-transition probability, an absorbing state does not emit any observation so that typically not all the rows of V sum to 1. Component i of the estimation of the current state $\hat{\vec{z}}_t$ is computed as $P(Z_t = j \,|\, X_{1:t-1} = x_{1:t-1}) = \frac{\sum_i w_{i,j}(x_{t-1})\alpha_i(t-1)}{\sum_k \alpha_k(t-1)}$.

As previously mentioned, in online conformance checking, most cases might not have reached completion. This means that monitoring their conformance does not give a full picture. Similar to previous work [16], we include the concept of *completeness* to indicate whether if the entire trace has been observed since the beginning.

7.2.4. Conformance metrics

Figure 7.6 graphically illustrate the two distinct aspects of conformance that our proposed technique measures. *Conformance* is between the current observation and the state estimation of the corresponding case. *Completeness* indicates whether if we are observing the complete trace by looking at previous observable units and the total injected distance [16]. Injected distance refers to the number of states that are skipped so that a running case can be brought from its last observed state to its updated state throughout its execution. This means that completeness is inversely correlated with the total amount of injected distance. For example, trace σ_0 in Figure 7.2 would have both perfect conformance and completeness since it corresponds to a complete model trace in the model in Figure 7.1. In contrary, trace σ_5 would yield a high conformance value but a low completeness since it corresponds to a partial model trace that is missing the prefix of at least two activities (NEW and FIN).

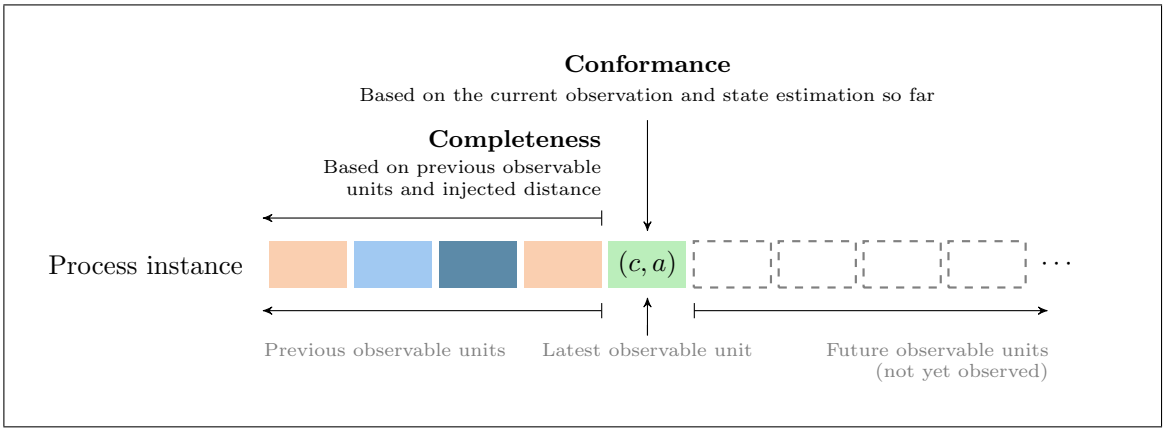


Figure 7.6. General idea of the two conformance indicators based on a running process instance: *conformance, completeness* (based on a similar diagram in [16])

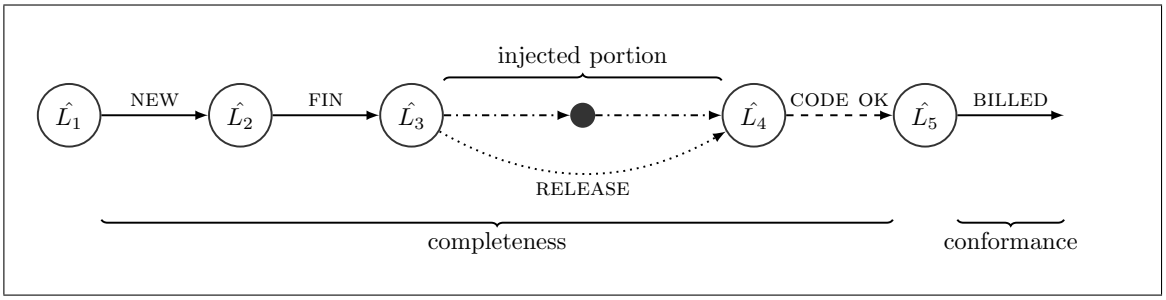


Figure 7.7. Metric breakdown projected onto the evaluation of trace σ_6 in Figure 7.4. Same as Figure 7.4, line style indicates various conformance explanations.

We refer to trace σ_6 of the running example in Figure 7.2 to provide some more intuition. Figure 7.7 enriches Figure 7.4 with a breakdown of the different conformance metrics. We can see that each state node is denoted by $\hat{Z}_i$ and corresponds to the state estimation at each time step. As before, suppose that we assume that all cases should start at the initial marking, the state estimation $\hat{Z}_1 = \vec{1}_{[i]}$ would concentrate all the probability mass on marking $[i]$. Similarly, $\hat{Z}_2 = \vec{1}_{[p_0]}$ and $\hat{Z}_3 = \vec{1}_{[p_8,p_9]}$. Due to the non-conforming event (RELEASE) and the following event (CODE OK), the state estimation $\hat{Z}_4$ concentrates all the probability mass on marking $[p_2]$. Inspecting the reachability graph of the model would tell us that the markings $[p_8, p_9]$ and $[p_2]$ are not adjacent and are separated by another node. This "injected distance" indicates that the observed sequence of events does

not correspond to a complete model trace. Moreover, for this particular state representation of markings, an event should not bring the corresponding case's state to a non-adjacent state, the total injected distance can be normalized and inverted as the completeness metric.

7.2.5. Algorithm for online processing

The procedure for online conformance computation is presented in Algorithm 2. The algorithm requires a stream of observable units (cf. Definition 2.13), the HMM-based model (cf. Definition 7.2), and a state distance matrix as input.

The algorithm has an infinite loop to process a stream of observations (lines 1 and 2). The whole procedure can be split into three phases: 1) updating the state estimation of a case upon a new event, 2) computing conformance, and 3) entry removals if the number of tracked cases is reaching maximum capacity.

For the first phase (lines 3 – 10), we update the discrete time step of the case, i.e., the case length, the forward probability, and the state estimation. Forward probability is computed with respect to each latent state. If the new event is the first observed event of the case, then the forward probability just corresponds to updating the initial distribution using the observation probability. Otherwise, we also need to account for the state-transition probability from each state of the previous forward probability. The update state estimation then corresponds to the normalized forward probability. For the second phase (lines 11 – 18), we compute three conformance metrics. The *conformance* of the observed event with respect to its estimated state is computed in line 11. Then, we use the modes of estimated previous and current state (which are categorical distributions) to update the *total injected distance* (line 12 – 17). We assume that in a conforming scenario, the two states should have a distance of 1 so that the observed event corresponds to the firing of a transition that progresses the previous marking to an adjacent marking in the reachability graph. To convert the total injected distance into a *completeness* metric, we assume that the sum of the total injected distance and the case length corresponds to the number of latent states

Algorithm 2: Online conformance computation

Input: S: stream of observable units
$M = (W_{x_1}, \ldots, W_{x_{|X|}}, W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V, V^d, \mathit{conf}, \pi)$: HMMConf
$D \in \mathbb{R}^{|Z| \times |Z|}$: state distance matrix

1 **forever do**
  // New caseid-activity pair from the stream
2 $(c, a) \leftarrow \mathit{observe}(S)$;
  // Phase 1: update forward probability $P(Z_t = i, X_{1:t} = x_{1:t})$
3 $\mathit{time}(c) \leftarrow \mathit{time}(c) + 1$; // $\mathit{time}(c) = 0$ if a is the first event
4 **for** $j \in Z$ **do**
5 **if** $time(c) = 0$ **then**
6 $\alpha^c_j(\mathit{time}(c)) \leftarrow \pi_j v_j(a)$; // c denotes the caseid
7 **else**
  // $a_{\mathit{time}(c)-1}$ is the activity observed at $\mathit{time}(c) - 1$
8 $\alpha^c_j(\mathit{time}(c)) \leftarrow \sum_{i \in Z} v_j(a) w_{i,j} (a_{\mathit{time}(c)-1}) \alpha^c_i(\mathit{time}(c) - 1)$;
9 **for** $i \in Z$ **do**
  // $P(Z_t = i \,|\, X_{1:t} = x_{1:t}) = \frac{P(Z_t = i, X_{1:t} = x_{1:t})}{\sum_{k \in Z} P(Z_t = k, X_{1:t} = x_{1:t})}$
10 $\mathit{state}(c, i) \leftarrow \frac{\alpha^c_i(\mathit{time}(c))}{\sum_{k \in Z} \alpha^c_k(\mathit{time}(c))}$;
  // Phase 2: compute online conformance values
11 $\mathit{conformance}(c) \leftarrow \mathit{conf}(\mathit{state}(c), a)$;
  // Modes are used to compute injected distance
12 **if** $\mathit{time}(c) = 1$ **then**
13 $\hat{z}_{\mathit{time}(c)-1} \leftarrow \mathrm{argmax}_{i \in Z}\ \pi_i$;
14 **else**
15 $\hat{z}_{\mathit{time}(c)-1} \leftarrow \mathrm{argmax}_{i \in Z}\ \alpha^c_i(\mathit{time}(c) - 1)$;
16 $\hat{z}_{\mathit{time}(c)} \leftarrow \mathrm{argmax}_{i \in Z}\ \alpha^c_i(\mathit{time}(c))$;
  // $\mathit{inj}(c) = 0$ if a is the case's first event
17 $\mathit{inj}(c) \leftarrow \mathit{inj}(c) + \max\{0, D_{\hat{z}_{\mathit{time}(c)-1}, \hat{z}_{\mathit{time}(c)}} - 1\}$;
18 $\mathit{completeness}(c) \leftarrow \frac{\mathit{time}(c)}{\mathit{inj}(c) + \mathit{time}(c)}$;
  // Phase 3: cleanup
19 **if** *size of* α *and state is close to max capacity* **then**
20 Remove entries of oldest cases ;

traveled across in the most likely latent state sequence so that completeness corresponds to the proportion of latent states that can be mapped to observations (line 18). The third phase of the algorithm (lines 19 – 20) removes the oldest entries due to the finite amount of memory to cater possibly an infinite number of cases.

Suitability for online settings. The computational complexity of the infinite loop is linear with respect to the stream size given the reference model as input. In phase 1, both the forward probability and state estimation corresponds to matrix operations of vectors and

matrices that have a fixed size given the reference model. This means they can be computed in constant time for each event. Similarly, all computations in phase 2 can be done in constant time given the reference model. Phase 3 can also be done in constant time using data structures like LinkedHashMaps. The space required by the algorithm is bounded by the maximum number of tracked cases. For each case, a vector of estimated state and several metric values are stored. Since processing an event takes a constant amount of time and space, the algorithm is suitable for online processing. Next, we turn to the task of computing and estimating the model parameters of the proposed technique.

7.3. Parameter computation and estimation

In the previous section, we presented the proposed technique and explained how online conformance checking can be performed. This section presents the parameter computation and estimation that are done offline on past data.

As previously presented, Figure 7.3 illustrates the dichotomy of the entire procedure where grayed boxes corresponds to data sources, dash lined boxes corresponds to parameter estimations, and solid lined boxes corresponds to other computations. In the following, we present the various details of the offline component, starting with the parameter estimation aspect. We note that this is for the presented instantiation of the proposed technique where we use Petri net markings to represent a case's state in the process. The conformance matrix that is used for the conformance function (cf. Definition 7.1) is computed by traversing the Petri net model.

Computation of conformance matrix. By traversing the reachability graph $G = (V, E)$, we compute a matrix $(c_{ij}) \in \mathbb{R}^{|V|\times|A|}$ so that $c_{m,a} = 1$ iff it is possible to observe activity a because either the corresponding transition is enabled at marking m or if there is a sequence of enabled invisible transitions whose firings would lead to a marking m' that enables the corresponding transition. Formally, $\forall_{m_i \in V}\ \forall_{t \in \{t' \in T | l(t') \in A\}} c_{m_i, l(t)} = 1$ iff $\exists_{j>i} \langle m_i, \ldots, m_j \rangle$ s.t. $(m_{j-1}, m_j) = t \wedge \forall_{i \leq q < j-1}\ l(m_q, m_{q+1}) = \tau$.

Computation of distance matrix. Similar to the computation of the conformance matrix, we traverse the reachability graph $G = (V, E)$ to compute a distance matrix $(d_{ij}) \in \mathbb{R}^{|V|\times|V|}$ so that d_{ij} corresponds to the shortest path distance between nodes v_i and v_j in G. Moreover, since invisible transitions are not observable in the event log, edges $e \in E$ that corresponds to invisible transitions have weight 0. Lastly, there can exist node pairs that do not have a directed path from one to the other. For these node pairs, we compute the shortest undirected path. Removing the edge direction from G yields a connected graph since all markings are reachable from the initial marking.

Parameters of conforming probability distributions. In the case where a case is perfectly fitting with respect to the Petri net model, the corresponding marking sequence is not hidden, i.e., the parameters can be directly estimated rather than in an iterative manner using the EM algorithm. Standard replay techniques can be used to yield compute the marking sequence. The remaining issue is therefore on the firing of invisible transitions which cannot be mapped to an observed event. For this, we assume that all firings of invisible transitions from the last recorded marking is related to the current observation; this gives a consistent interpretation even in the case of requiring the firing of invisible transitions at the initial marking.

As such, both the state-transition and emission probability distributions are categorical distributions that describes the probability of transitioning to different next states given the current state and the probability of observing a particular activity given the current state respectively. The parameters can then be estimated by normalizing the respective counts from the replay of the cases in the training sets.

However, it is possible that not all of the modeled behavior has been observed in the training set. We take a Bayesian approach to incorporate the knowledge of all possible modeled behavior captured from the Petri net model into the distributions. Given that both distributions are categorical distributions, modeled behavior can be added as pseudo counts by using a Dirichlet prior [52]. To accumulate the pseudo counts, we traverse the reachability graph in the same manner as for the conformance matrix. For simplicity

and computation speed, rather than using the full posterior distribution of the parameters, the expected values are used as point estimates of the parameters. This corresponds to normalizing the parameters of the Dirichlet posterior.

Parameters of non-conforming probability distributions. Following the parameter estimation of the conforming probability distributions, we now turn to the more difficult task of estimating the parameters of the non-conforming state-transition and emission probability distributions. This corresponds to finding parameters that maximize the likelihood of the observations given the parameters:

$$W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V^d = \text{argmax}_{W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V^d} P(x_{1:T}; W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V^d)$$

This is difficult to directly optimize because both the parameters and the corresponding latent states of the observations are free parameters with dependence. Therefore, rather than direct optimization, the EM algorithm [23] is used. The EM algorithm alternates between estimating the latent states (Expectation step) and the matrix parameters (Maximization step) until the convergence threshold is met.

At the Expectation step, we fix the current parameter estimates and together with the observations, we compute the conditional probability of each observation being at the different latent states as $Q(z_{1:T}) = P(z_{1:T} \mid x_{1:T}; W_{x_1}, \ldots, W_{x_{|X|}}, V, W^d_{x_1}, \ldots, W^d_{x_{|X|}}, V^d)$. Then, at the Maximization step, we find new parameter estimates that maximize the conditional expected log likelihood of both the observations and their latent states. Similar to a conventional HMM, we can set up and derive closed form updates for the parameter

estimates as follows:

$$\begin{aligned}\log P(x_{1:T}) &= \sum_{z_{1:T}} \log P(x_{1:T}, z_{1:T}) \\ &\geq \sum_{z_{1:T}} Q(z_{1:T}) \log \left[\frac{P(x_{1:T}, z_{1:T})}{Q(z_{1:T})} \right] \\ &= \sum_{z_{1:T}} Q(z_{1:T}) \left[\log \pi_{z_1} + \underbrace{\sum_{t=2}^{T} \log w_{z_{t-1}, z_t}(x_{t-1})}_{1} + \underbrace{\sum_{t=1}^{T} \log v_{z_t}(x_t)}_{2} \right] \end{aligned} \tag{7.3}$$

We can separately estimate the two variables of interest by focusing on the labeled parts of Equation 7.3:

(i) State-transition probability matrix of deviating behavior W_a^d

(ii) Observation probability matrix of deviating behavior V^d

$$W^d_{a,i,j} = \frac{\sum_{t=2}^{T} \alpha_i(t-1) w_{i,j}(z_{t-1}) v_j(x_t) \beta_j(t)\ 1\{x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}}{\sum_{j=1}^{|Z|} \sum_{t=2}^{T} \alpha_i(t-1) w_{i,j}(z_{t-1}) v_i(x_t) \beta_j(t)\ 1\{x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}}$$

$$V^d_{j,a} = \frac{\sum_{t=1}^{T} 1\{x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} \alpha_j(t) \beta_j(t)}{\sum_{t=1}^{T} 1\{\textit{conf}(\vec{z_t}, x_t) < 1\} \alpha_j(t) \beta_j(t)}$$

$\beta_j(t) = P(X_{t+1:T} = x_{t+1:T} \mid Z_t = j, X_t = x_t)$ is the backward probability for state j at time t and is computed as $\beta_j(t) = \sum_{k \in Z} v_k(x_{t+1}) w_{j,k}(x_t) \beta_k(x_{t+1})$ with base case $\beta_j(T-1) = \sum_{k \in Z} v_k(x_T) w_{j,k}(x_{T-1})$.

Next, we present the experimental evaluation of the proposed technique where we performed a stress test and correlation test on existing dataset for comparability with state of the art techniques.

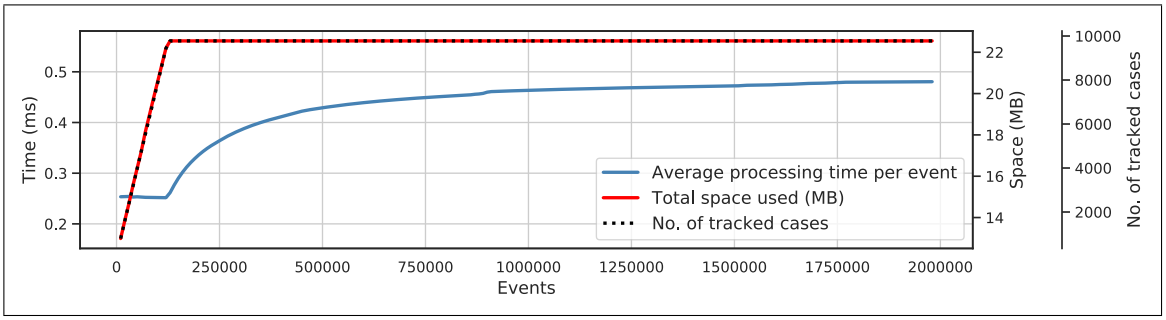


Figure 7.8. Performance during a stress test of ∼2 million events (see colored version online)

7.4. Experimental evaluation

For the sake of space and scope, we focus the evaluation on the conformance checking results of the proposed model rather than its predictive capability. The proposed approach is implemented in Python and can be found in the GitHub repository[2], along with further detail for reproduction. The experimental results are open and publicly available [3].

7.4.1. Stress test

Similar to [16], we performed a stress test of our approach using the dataset from the mentioned work so that the results are comparable. However, preprocessing was necessary to filter out noisy events with randomly generated activity names. After filtering, the event stream has 1,985,744 events[4]. The test was performed on a standard machine, equipped with Python 3.6, an Intel Core i7-4700MQ 2.40GHz CPU and 12GB of RAM. We allowed the model to track 10,000 cases at most.

Figure 7.8 presents the results. Space is measured as the object size of the model. We can see that space is directly correlated with the number of cases after adding on a fixed space for the parameters. Both the number of cases and total space used reach a peak

[2] https://github.com/jwllee/HMMConf/tree/Computing2019
[3] DOI will be requested after the acceptance of the manuscript. Meanwhile, they can be accessed at https://drive.google.com/open?id=1w_Lt6aPmbwHFgP6BPpy3Mel0_AGhGDK7
[4] Models and streams available in https://doi.org/10.5281/zenodo.1194057

and remain stable at around 125k events when all 9,778 cases are tracked. Since the state transition probability is not involved for the first event of a case, the average processing time is particularly low for the first 125k events. Then, time eventually stabilizes at ∼0.49 ms after 1M events. This test demonstrates that the model can sustain a high load of events. As expected, the results suggest that both processing time and memory usage are non-increasing with respect to the number of events after reaching stability. However, it is significantly slower than the average processing time of below 0.009 ms/event reported in [16].

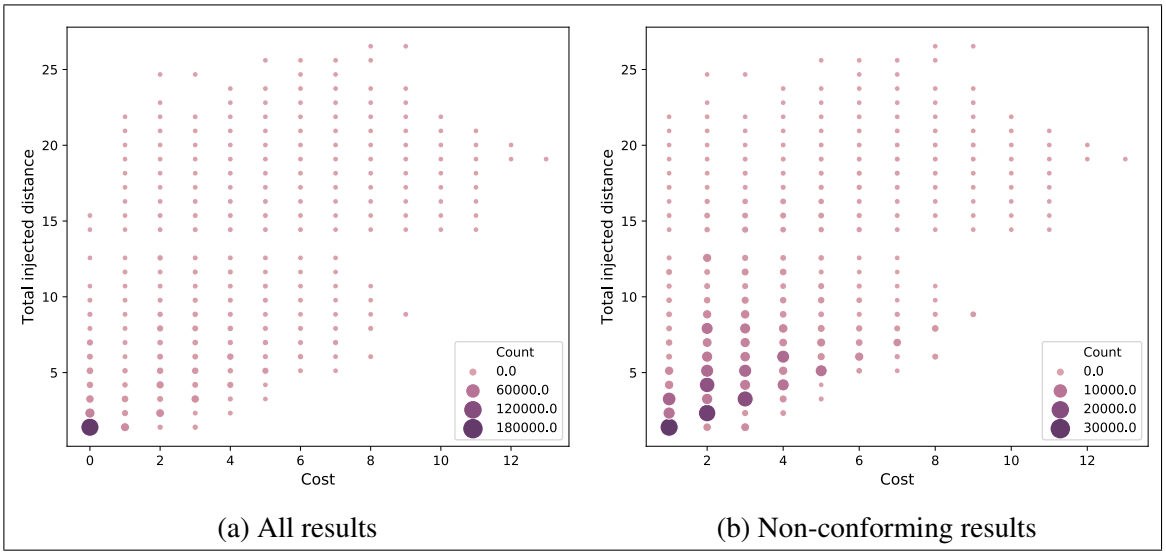
7.4.2. Correlation with alternative conformance metrics

In this section, the proposed technique is compared with an alternative cost-based prefix alignment technique described in [78]. As the cost-based metric computed by alignment is a more informative and well-established technique, correlation between the two techniques would suggest that the proposed technique also reflects conformance as understood in the literature.

Similar to before, we make use of the dataset generated for the correlation test in [16]. Given that the prefix alignment results were not available in the open source dataset, we implemented prefix alignments using the Alignment package[5] in ProM6 [86]. The alignments are then computed using the standard cost function [4]. For each Petri net model, a 5-fold cross validation is performed on all the traces so that the mean measurement is taken for each event. Moreover, for the EM parameter estimation, we set the convergence condition as a tolerance threshold of 5 in log probability difference or a maximum of 10 iterations. Since we know that the traces can be non-conforming, an epsilon of $\sim$ 1e-5 (0.001%) is added to all states of the initial distribution.

We compare the two techniques under the context of all results and only non-conforming results as determined by alignment costs. Overall, we find that total injected distance is

[5] `https://svn.win.tue.nl/trac/prom/browser/Packages/Alignment`



(a) All results (b) Non-conforming results

Figure 7.9. Bubble plots of total injection distance (with epsilon mass at initial distribution) versus incremental alignment costs

conceptually more similar to alignment costs. Figure 7.9 presents bubble plots where total injected distances are binned so that the y-axis values are of the intervals' mid value. Moreover, Figure 7.10a presents the Spearman's rank correlation coefficient (ρ-value) between costs and the proposed metrics. Given that conformance computed by our approach is between the estimated state and one event, we use the mean conformance for comparison. This means that for a case of length k, the mean conformance is computed from k values.

As shown in Figure 7.9a, the dataset is predominantly conforming. In fact, as we will later show in the confusion matrix analysis, according to the prefix alignment technique, only $\sim 20\%$ of the case prefixes are non-conforming. Referring to Figure 7.10a, there is a ρ-value of 0.697 between total injected distance and cost. The moderate positive correlation between the two is expected since higher costs imply that a larger number of consecutive latent states are likely to be not adjacent in the reachability graph. In addition, the correlations between costs and mean conformance (-0.470) and completeness (-0.712) are also within expectations. For the non-conforming results, visually, Figure 7.9b suggests that higher costs correlates with larger total injected distance and this is supported by

(a) Spearman correlation

	Spearman correlation coefficient[a]	
	All	Non-conforming
Mean conformance	-0.470	0.252
Total injected distance	0.697	0.665
Completeness	-0.712	-0.519

[a]All results are statistically significant with two-sided p-value <0.001

(b) Confusion matrix

	Non-conforming	conform[a]
cost > 0	450794	17763
cost $= 0$	333356	2093647

[a]conformance >0.99 and total injection distance is 0

Figure 7.10. Statistics comparing prefix alignment costs and three metrics

a ρ-value of 0.665. We observe a similar result with the moderate negative correlation between costs and completeness (-0.519). There is a low positive correlation between mean conformance and costs (0.252). This is surprising as one would expect conformance to be negatively correlated with costs. However, it is likely that the two metrics are simply measuring different conformance qualities. The proposed approach's conformance metric measures whether if the observed event is conforming with respect to the local position of the case while costs takes a global perspective in measuring the cost of aligning the observed trace in an optimal manner. In comparison to the previous work on a behavioral pattern based approach [16] which yielded a ρ-value of -0.954 for the whole dataset and a ρ-value of -0.295 for non-conforming results, the results suggest that our approach yields conformance results that are closer to those provided by prefix alignments than the behavioral pattern based approach under non-conforming scenarios. One of the reasons for the lower correlation when considering the whole dateset is due to the differences at the lower cost region. This is likely to be due to the fact that our approach handles warm start scenarios by quickly orienting the case at the corresponding location in the model rather than classifying events as log moves.

An important scenario in conformance checking is the classification of whether a case is conforming or not. For this, alignment can be treated as the ground truth. Figure 7.10b shows the confusion matrix of our approach's performance. A case is deemed to be conforming if the conformance of the current event >0.99 (to account for float imprecision) and has a total injected distance of 0. We find that our approach has a high precision of

0.992 and recall is at 0.863. This gives a F1-score of 0.923 and is much better than 0.838, the F1-score of a stratified dummy classifier.

In conclusion, we find that our approach yields results that correlate with prefix alignments. In particular, the total injected distance metric has a moderate correlation with alignment costs under both conforming and non-conforming scenarios. Moreover, differences between the two techniques can be adequately explained by the differences in treatment of both conforming and non-conforming scenarios, as well as the limitations of our approach.

7.5. Real-life dataset evaluation

We also perform an evaluation on a real-life dataset - hospital billing event log [45]. One focus is to illustrate its applicability in the current real life context where often times there is no available normative process model corresponding to the event data.

Preprocessing and model construction. Given that the event log spans over four years from 2012 to 2016, during which it can be shown that the process went through concept drifts, we filtered the event log to only contain cases that started at 01-01-2013 and after. Moreover, since our focus is on tracking the conformance of cases over the course of their executions, it is easier to highlight this aspect by looking at cases that have more than just a few events. We perform our final filter to only include cases that have 10 or more events. The final event log contains 2992 cases and 630 trace variants. As previously mentioned, it can be difficult to find a normative process model in the current real life context. One possible solution is to construct a process model from the observed data as a proxy of the underlying process. Here we assume that the underlying process is composed of the most frequent trace variants. Figure 7.12 shows that most of the cases belong to only a few trace variants. In fact, there are only 10 trace variants with more than 50 cases and in total there are 1717 cases associated with these 10 trace variants, i.e., they make up for more than 50% of the event log. We then created a handmade model that permits the 10 trace

variants as shown in Figure 7.11. Overall, there are two main parts to the process: the billing goes through an approval process to yield a code before getting billed.

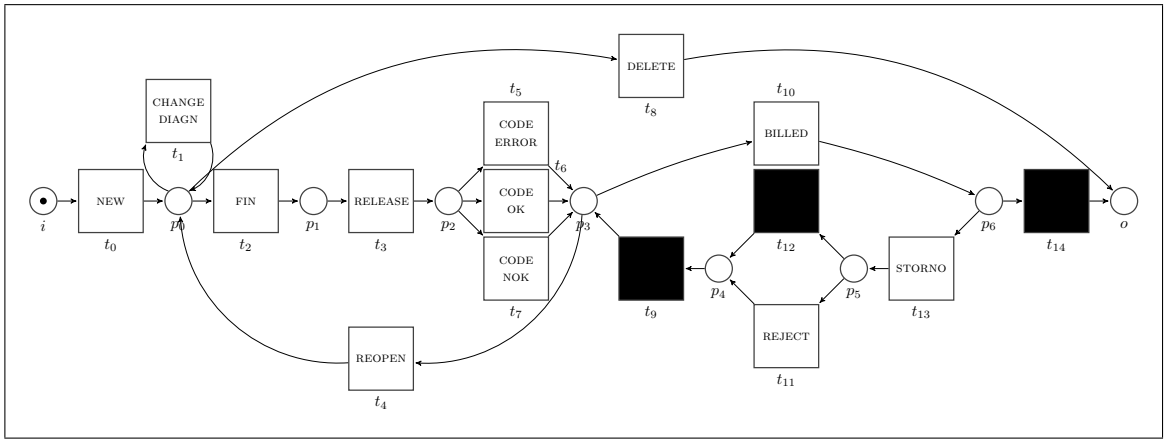


Figure 7.11. Petri net model extracted from 10 most frequent trace variants

Experiment setup. We used a k-fold cross validation approach to evaluate the model where $k = 5$ and stratified sampling is used over the variant category of the cases so that the training set has a similar distribution as the overall dataset. For the training of the proposed model, a maximum of 10 iterations is set for the EM algorithm.

One interesting challenge is that the process model only has 12 visible transitions and does not include all 17 possible activities since some of the activities are not included in

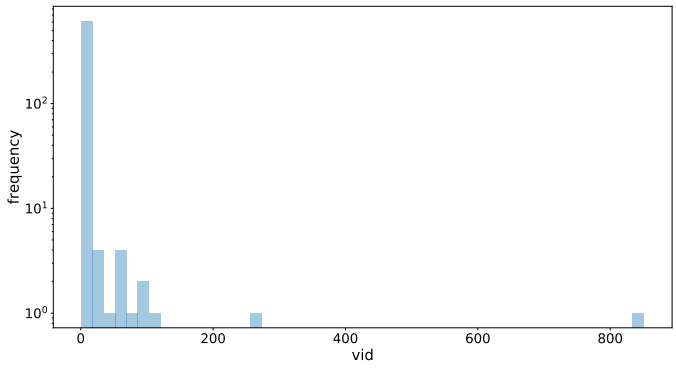
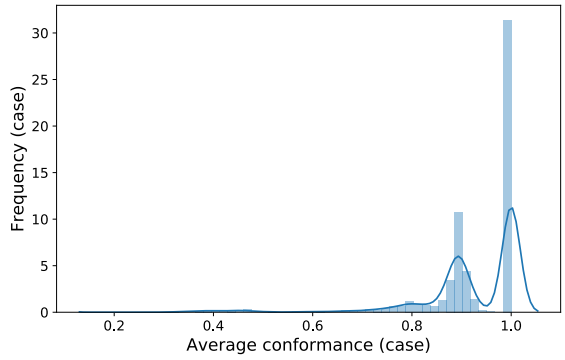


Figure 7.12. Distribution plot showing concentration of cases on a few trace variants

any of the 10 trace variants. We assumed that in real life we are able to know the set of possible observable activities beforehand so that these unmodeled activities can be incorporated into the observation variable set of the probability distributions and conformance matrix. Note that an even weaker assumption of not knowing the set of possible activities can be taken by adding a wild card activity into the observation variable set so that all unmapped activities are mapped to this wild card activity.

Result analysis. Overall, we find that the data have high conformance. This is expected since the model captures $> 50\%$ of the observed behavior on a trace level. Inspecting the results on an activity level identifies specific conformance problems.



(a) Distribution plot of average conformance per case

Fold no.	Conformance		Total injected distance		Completeness	
	mean	std	mean	std	mean	std
1	0.930	0.233	0.172	0.528	0.985	0.045
2	0.947	0.205	0.119	0.420	0.990	0.035
3	0.951	0.198	0.108	0.389	0.991	0.033
4	0.834	0.330	0.475	1.153	0.966	0.075
5	0.944	0.211	0.145	0.480	0.988	0.040
All	0.901	0.269	0.267	0.817	0.980	0.057

(b) Test set statistics of conformance metrics

Figure 7.13. Experiment results on case level

Table 7.13b presents the mean and standard deviations of the test set conformance results. We can see that with the exception of the fourth test fold to some extent, all of the results are quite similar. This is expected since a stratified sampling approach was taken to create the cross validation data partition. The results show that generally the data is conforming and the injected distance is quite low. However, the standard deviation is somewhat high. Figure 7.13a shows the distribution plot of the average conformance per case. It shows two pronounced peaks: one at 1.0 and the other at 0.9. In fact, 1534 cases

resulted with an avg. conformance of large or equal to 0.99 and 1173 cases resulted with an avg. conformance within the interval of $[0.80, 0.99)$.

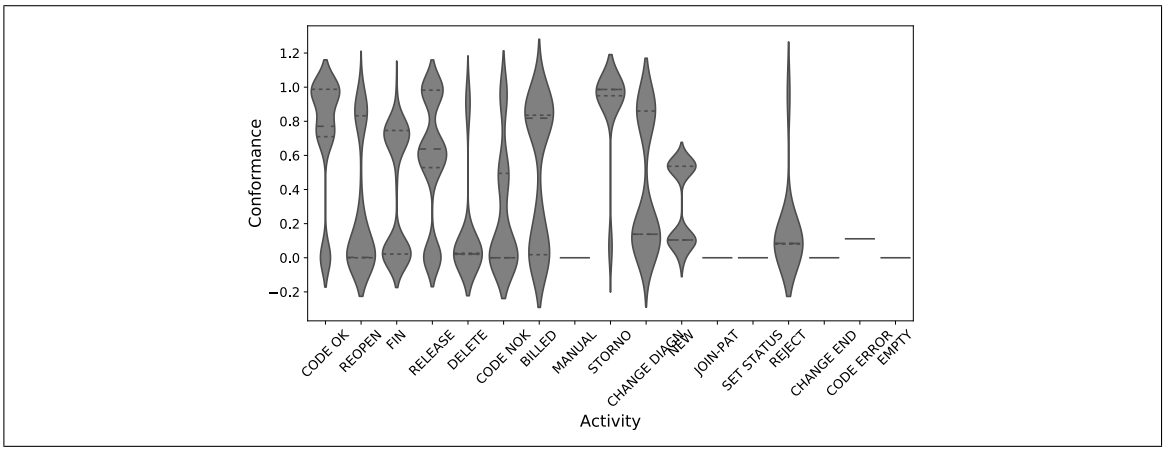
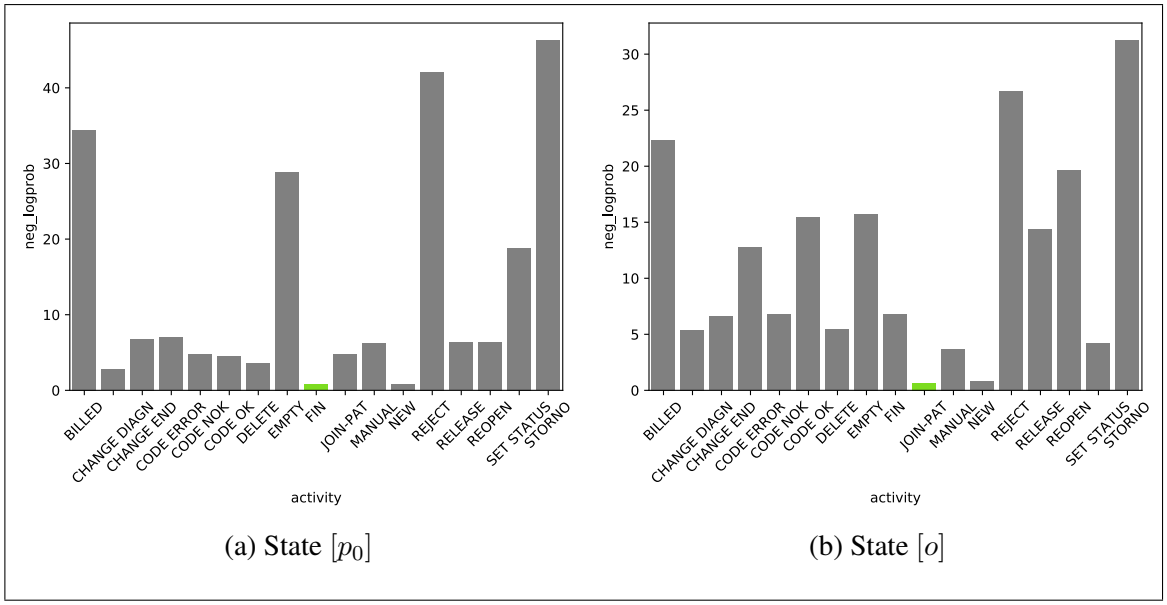


Figure 7.14. Violin plots of the conformance per activity for non-conforming events

Next, we analyze the results from the activity perspective. Figure 7.14 shows a violin plot of events which have a conformance value lower than 0.99, i.e., non-conforming events. It shows that the conformance values are distributed with multiple modes. This is actually due to the nature of the conformance issues in the data. For most of the non-conforming cases, the problem is in having "extra" activity executions that are not modeled. This means that either an activity is observed while the case is at the "wrong" location within the process or that a series of previous non-conforming observations had caused uncertainty in the state estimation but most of the probability mass is at the "correct" location for the current observation. Analyzing the specific shapes of the plots can even point out the particular conformance problem. For example, looking at the plot shapes of the activities CODE OK and CODE NOK shows that for the non-conforming observations, observations of CODE OK generally have high conformance and observations of CODE NOK generally have low conformance. Inspecting the cases in the event log shows that there are 77 cases where there is a string of one or more events with CODE

NOK that ends with CODE OK before moving onto a different part of the process. In contrary, there is only 1 case where an event of CODE OK is followed by CODE NOK and even in this case, the event corresponding to CODE NOK ultimately ends with CODE OK.

One possible explanation is that there can be problems in obtaining a valid declaration code but that trying multiple times can ultimately lead to a successful declaration code.



(a) State $[p_0]$ (b) State $[o]$

Figure 7.15. Non-conforming emission probability distributions at states $[p_0]$ and $[o]$

Finally, we discuss about the unmodeled activities. Specifically, we look at the activity JOIN-PAT which joins multiple billings of the same patient. Since there are only 57 cases with this activity, one can inspect the event data to find that the activity mostly occur right after the billing package is closed or at the end of the case. Inspecting the non-conforming emission probability distribution suggests the same observation. Figure 7.15a and Figure 7.15b shows the emission probability distributions at the states $[p_0]$ and $[o]$ respectively. The probability values are presented as negative log probability so that activities with lower values are more probable. For both states, we see that the activity JOIN-PAT have relatively low values. In fact, JOIN-PAT is the most probable observation at state $[o]$. The coincidence between the observation from the event log and the emission

probability distributions suggest that the proposed technique was able to learn correctly from the training data.

7.6. Related work

There are many offline conformance checking techniques, e.g., alignment-based techniques [4, 63, 75, 54, 1, 6, 63, 77, 61], behavioral profile techniques [92], token replay based techniques [58, 82], and HMM techniques [59]. This work differs from [59] in that the state-space of the model is used as the latent variable set and both the previous state and observation are used to estimate the next state. In addition, the EM algorithm is used to estimate the state-transition and observation probabilities under non-conforming scenarios. For online conformance checking, we discuss three recent works. Prefix alignments are presented in [78] to provide alignment explanations for possibly ongoing cases. However, alignment computations can take a long time and also the technique cannot handle warm start scenarios. Another approach is to pre-compute possible deviations on top of the model behavior [15]. There is some similarity between the proposed approach and [15] in that we try to keep track of the state of a running case. But rather than deciding how non-conformance should be handled beforehand, the EM algorithm is used to estimate the necessary parameters. Lastly, [16] transforms an event stream into a stream of behavioral patterns and checks whether if the observed patterns are conforming. This approach takes a strong abstraction to trade for run time efficiency. It also proposed a breakdown of conformance in an online context into three aspects: conformance, completeness, and confidence. This idea was brought into our approach as: conformance and total injected distance.

7.7. Conclusion and future work

This chapter presents an approach to perform conformance checking on a stream of events against a reference model. The approach alternates between updating the state estimation of a running case and computing its conformance with respect to the updated state.

To model the behavior of a process under both conforming and non-conforming scenarios, the approach modifies a conventional HMM so that both the state estimation and observations are used to estimate the next state. Similar to a recent work [16], to measure different aspects of conformance, the approach includes three different metrics: conformance, total injected distance, and completeness. The proposed approach is implemented as a Python package and has been verified through a stress test, a comparison with prefix alignments, and a real-life dataset. As future work we plan to address the identified limitations as well as to further develop the approach. This includes investigating ways to abstract from using markings as latent states for efficiency and other constructs to model behavior in the presence of non-conformance, e.g., decomposition techniques [65].

Part V

Closure

8. Conclusions

In this section, we present a summary of the contributions of this thesis, challenges and directions for future work.

8.1. Summary of contributions

The main theme of this thesis is on conformance checking. In the introduction, we identified four challenges arising from different data dimensions: *(1) conformance checking algorithms can be computationally expensive due to the large possibly infinite state-space of process models*, *(2) decomposition techniques is a promising solution to reduce computational complexity since it can break down a large conformance checking problem into smaller ones, but existing decomposition techniques tackle the decisional problem of whether if a log trace is conforming or give best effort results that are a lower bound of the exact result*, *(3) given the difficulty of conformance checking, there can be many algorithms that do the same task but with different characteristics and advantages at different scenarios, this makes it difficult for the end user to choose between them without having expert knowledge of the algorithms*, and *given the volume and velocity at which event data comes in, organizations may not store data for offline analysis and have to resort to online techniques, online conformance checking brings in challenges that are not present in offline scenarios.* In this thesis, we have presented results that address the identified challenges:

- **Decomposition techniques.** Chapter 3 presents the total border agreement condition as a sufficient condition to merge optimal sub-alignments so that the merged alignment has the same misalignment costs as an optimal alignment computed under the monolithic approach. The condition requires that alignment moves related to border activities are synchronized across all sub-alignments. Chapter 4 makes use of this merging condition to propose a divide and conquer framework for computing alignments. The proposed framework iteratively performs decomposed replay

to check for the merging condition and recomposes on problematic border activities to resolve border conflicts in the following iteration. Moreover, the framework allows the user to configure the level of accuracy in the conformance results so that they can choose the balance between result accuracy and computation time. Experimental results show that there can be some traces that can take many iterations to reach the needed merging condition. This forms a bottleneck for the proposed recomposing conformance checking framework. Chapter 5 addresses this problem by investigating different recomposition heuristics that can encourage the merging condition.

- **Algorithm selection.** Chapter 6 applies machine learning to create an oracle that can help user use the best algorithm depending on their data. Specifically, the chapter investigates the problem of choosing when to use decomposition techniques for computing alignments depending on the input data.
- **Online conformance checking.** Chapter 7 turns to the challenge of performing conformance checking in an online setting. Here the focus is on the challenge of having to balance between making sense at the process level as the case reaches completion and putting emphasis on the current information at the same time. The chapter proposes a Hidden Markov Model (HMM) based framework that alternates between estimating the current position of the running case in the scope of the process and computing the conformance between the observed and modeled behavior using the estimated case position.

8.2. Challenges and future work

In this section we list some of the possible future research lines:

- **Outside of alignments.** A large part of this thesis has been focused on improving the computational performance of alignment techniques by using decomposition. However, as of current, alignments is no longer the only conformance result that are computationally intensive. For example, good metrics for the precision dimension

inevitably have to explore a large number of reachable model states to compare the modeled behavior with the observed behavior. Decomposition techniques can be applied to partition the process for such exploration.

- **Algorithm selection.** Often times proposed techniques are focused on specific issues, e.g., the recomposing framework is focused on large and complex processes. This means that the end user needs to have the expertise to decide the appropriate technique for the conformance checking task at hand. It would be desirable to have an oracle that can abstract this for general tasks such as alignment computation. In Chapter 6, we applied machine learning to abstract the decision between the monolithic or decomposed approach for alignment computation to achieve best computational performance. Finding a good oracle is difficult since the number of possible problem configurations is often infinite. Furthermore, in the research space, techniques are often improved. This makes it difficult to have an oracle that is consistent with the current state of the art.
- **Model repair.** Computing conformance is only the first part of the process. After finding out the conformance issues, actions need to be taken to address and resolve the identified issues. However, the challenge is in finding the appropriate balance between all the different conformance dimensions while maintaining certain desirable properties of the repaired model such as soundness.
- **Online conformance checking.** Part of the challenge in online conformance checking is the extra computation complexity requirement imposed by stream processing. This means that often one has to resort to approximations. Moreover, the uncertainty of how a case will unfold means that results of an online conformance checking technique can be unstable.

8.3. Acknowledgement

This work is supported by *CONICYT-PCHA / Doctorado Nacional / 2017-21170612*.

REFERENCES

[1] H. Van Der Aa, H. Leopold, and H. Reijers. Efficient process conformance checking on the basis of uncertain event-to-activity mappings. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019.

[2] Ahmed Abbasi, Suprateek Sarker, and Roger H. L. Chiang. Big Data Research in Information Systems: Toward an Inclusive Research Agenda. *J. AIS*, 17(2), 2016.

[3] Giovanni Acampora, Autilia Vitiello, Bruno N. Di Stefano, Wil M. P. van der Aalst, Christian W. Günther, and Eric Verbeek. IEEE 1849: The XES standard: The second IEEE standard sponsored by IEEE computational intelligence society [society briefs]. *IEEE Comp. Int. Mag.*, 12(2):4–8, 2017.

[4] Arya Adriansyah. *Aligning Observed and Modeled Behavior*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[5] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 55–64, 2011.

[6] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Diagnosing behavioral differences between business process models: An approach based on event structures. *Inf. Syst.*, 56:304–325, 2016.

[7] Adriano Augusto, Abel Armas-Cervantes, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Daniel Reißner. Abstract-and-compare: A family of scalable precision measures for automated process discovery. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, volume 11080 of *Lecture Notes in Computer Science*, pages 158–175. Springer, 2018.

[8] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artif. Intell.*, 237:41–58, 2016.

[9] Vincent Bloemen, Sebastiaan J. van Zelst, Wil M. P. van der Aalst, Boudewijn F. van Dongen, and Jaco van de Pol. Maximizing synchronization for aligning observed and modelled behaviour. In *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, pages 233–249, 2018.

[10] Carmen Bratosin, Natalia Sidorova, and Wil M. P. van der Aalst. Distributed genetic process mining. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8. IEEE, 2010.

[11] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[12] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Wadsworth Statistics, 1984.

[13] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[14] Andrea Burattin. PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings. *CoRR*, abs/1506.08415, 2015.

[15] Andrea Burattin and Josep Carmona. A framework for online conformance checking. In *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain*, pages 165–177, 2017.

[16] Andrea Burattin, Sebastiaan J. van Zelst, Abel Armas-Cervantes, Boudewijn F. van Dongen, and Josep Carmona. Online conformance checking using behavioural patterns. In *Business Process Management - 16th International Conference, BPM 2018,*

Sydney, NSW, Australia, Proceedings, pages 250–267, 2018.

[17] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[18] Thomas Chatain and Josep Carmona. Anti-alignments in Conformance Checking - The Dark Side of Process Models. In *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, pages 240–258, 2016.

[19] Massimiliano de Leoni and Andrea Marrella. Aligning real process executions and prescriptive process models through automated planning. *Expert Syst. Appl.*, 82:162–183, 2017.

[20] Massimiliano de Leoni, Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzzocrea, and Timos K. Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, volume 8841 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2014.

[21] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

[22] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf. Syst.*, 37(7):654–676, 2012.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[24] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[25] Dirk Fahland, Massimiliano de Leoni, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance Checking of Interacting Processes with Overlapping Instances. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*, pages 345–361. Springer, 2011.

[26] Luciano García-Bañuelos, Nick van Beest, Marlon Dumas, Marcello La Rosa, and Willem Mertens. Complete and interpretable conformance checking of business processes. *IEEE Trans. Software Eng.*, 44(3):262–290, 2018.

[27] D. Gaurav, J. K. P. Singh Yadav, R. K. Kaliyar, and A. Goyal. An outline on big data and big data analytics. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 74–79, Oct 2018.

[28] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[29] Mieke Julie Jans, Michael G. Alles, and Miklos A. Vasarhelyi. The case for process mining in auditing: Sources of value added and areas of application. *International Journal of Accounting Information Systems*, 14(1):1–20, 2013.

[30] Toon Jouck and Benoît Depaire. Ptandloggenerator: A generator for artificial event data. In *BPM (Demos)*, volume 1789 of *CEUR Workshop Proceedings*, pages 23–27. CEUR-WS.org, 2016.

[31] Lars Kotthoff, Ian P. Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Commun.*, 25(3):257–270, 2012.

[32] Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards Understanding Process Modeling – the Case of the BPM Academic Initiative. In *International Workshop on Business Process Modeling Notation*, pages 44–58. Springer, 2011.

[33] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

[34] Wai Lam Jonathan Lee. Advancing Decomposed Conformance Checking in Process Mining. In *Business Process Management Doctoral Consortium, Sydney, Australia, September 09, 2018.*, 2018.

[35] Wai Lam Jonathan Lee, Andrea Burattin, Jorge Munoz-Gama, and Marcos Sepúveda. Orientation and conformance: A HMM-based approach to online conformance checking. *Information System (under review)*, 2019.

[36] Wai Lam Jonathan Lee, Jorge Munoz-Gama, H. M. W. Verbeek, Wil M. P. van der Aalst, and Marcos Sepúlveda. Improving Merging Conditions for Recomposing Conformance Checking. In *Business Process Management Workshops - BPM 2018 International Workshops, Sydney, Australia, September 10, 2018. Revised Papers*, 2018.

[37] Wai Lam Jonathan Lee, H. M. W. Verbeek, Jorge Munoz-Gama, Wil M. P. van der Aalst, and Marcos Sepúlveda. Replay using recomposition: Alignment-based conformance checking in the large. In *Proceedings of the BPM Demo Track and BPM Dissertation Award, Barcelona, Spain, September 13, 2017.*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[38] Wai Lam Jonathan Lee, HMW Verbeek, Jorge Munoz-Gama, Wil MP van der Aalst, and Marcos Sepúlveda. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Information Sciences*, 2018.

[39] Lee, W.L.J. (Jonathan), Verbeek, H.M.W. (Eric), Munoz-Gama, J. (Jorge), Van Der Aalst, W.M.P. (Wil), and Seplveda, M. (Marcos). Recomposing conformance (ins 2018), 2018.

[40] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - A constructive approach. In José Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2013.

[41] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery with guarantees. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Proceedings*, volume 214 of *Lecture Notes in Business Information Processing*, pages 85–101. Springer, 2015.

[42] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery and conformance checking. *Software and System Modeling*, 17(2):599–631, 2018.

[43] Yujian Li. Hidden markov models with states depending on observations. *Pattern Recognition Letters*, 26(7):977–984, 2005.

[44] María Teresa Gómez López, Diana Borrego, Josep Carmona, and Rafael M. Gasca. Computing alignments with constraint programming: The acyclic case. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016, Torun, Poland, June 20-21, 2016.*, pages 96–110, 2016.

[45] Mannhardt, F. (Felix). Hospital billing - event log, 2017.

[46] Mannhardt, F. (Felix). *Multi-perspective Process Mining*. PhD thesis, Technische Universiteit Eindhoven, 2018.

[47] Jorge Munoz-Gama. *Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes*. Springer, 2016.

[48] Jorge Munoz-Gama. Conformance checking. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019.

[49] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Hierarchical Conformance Checking of Process Models Based on Event Logs. In José Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 291–310. Springer, 2013.

[50] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Single-Entry Single-Exit decomposed conformance checking. *Inf. Syst.*, 46:102–122, 2014.

[51] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[52] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[53] Artem Polyvyanyy, Andreas Solti, Matthias Weidlich, Claudio Di Ciccio, and Jan Mendling. Monotone precision and recall measures for comparing executions and specifications of dynamic systems. *CoRR*, abs/1812.07334, 2018.

[54] Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Abel Armas-Cervantes. Scalable conformance checking of business processes. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*, pages 607–627, 2017.

[55] Joel Ribeiro, Josep Carmona, Mustafa Misir, and Michèle Sebag. A Recommender System for Process Discovery. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 67–83, 2014.

[56] Anne Rozinat. *Process Mining: Conformance and Extension*. PhD thesis, Technische Universiteit Eindhoven, 2010.

[57] Anne Rozinat and Wil M. P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, volume 3812, pages 163–176, 2005.

[58] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[59] Anne Rozinat, Manuela M. Veloso, and Wil M. P. van der Aalst. Using Hidden Markov Models to Evaluate the Quality of Discovered Process Models. *BPM Center*

Report, BPM-08-10, 2008.

[60] Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil M. P. van der Aalst. The imprecisions of precision measures in process mining. *Inf. Process. Lett.*, 135:1–8, 2018.

[61] Farbod Taymouri and Josep Carmona. A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models. In *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, pages 197–214, 2016.

[62] Farbod Taymouri and Josep Carmona. Model and Event Log Reductions to Boost the Computation of Alignments. In *SIMPDA 2016, Graz, Austria, December 15-16, 2016.*, pages 50–62, 2016.

[63] Farbod Taymouri and Josep Carmona. An evolutionary technique to approximate multiple optimal alignments. In *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, pages 215–232, 2018.

[64] Wil M. P. van der Aalst. Decomposing Process Mining Problems Using Passages. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*, volume 7347 of *Lecture Notes in Computer Science*, pages 72–91. Springer, 2012.

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

[66] Wil M. P. van der Aalst. *Process Mining - Data Science in Action*. Springer, 2016.

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

[68] Wil M. P. van der Aalst, Alfredo Bolt, and Sebastiaan J. van Zelst. Rapidprom: Mine your processes and not just your data. *CoRR*, abs/1703.03740, 2017.

[69] Wil M. P. van der Aalst, Kees M. van Hee, Jan Martijn E. M. van der Werf, Akhil Kumar, and Marc Verdonk. Conceptual model for online auditing. *Decision Support Systems*, 50(3):636–647, 2011.

[70] Wil M. P. van der Aalst, Kees M. van Hee, Jan Martijn E. M. van der Werf, and Marc Verdonk. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *IEEE Computer*, 43(3):90–93, 2010.

[71] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[72] B. F. van Dongen. BPI Challenge 2012, 2012.

[73] B. F. van Dongen. BPI Challenge 2017, 2017.

[74] B. F. van Dongen and F. Borchert. BPI Challenge 2018, 2018.

[75] Boudewijn F. van Dongen. Efficiently computing alignments - using the extended marking equation. In *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, pages 197–214, 2018.

[76] Boudewijn F. van Dongen, Josep Carmona, and Thomas Chatain. A unified approach for measuring precision and generalization based on anti-alignments. In Marcello La Rosa, Peter Loos, and Oscar Pastor, editors, *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2016.

[77] Boudewijn F. van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. Aligning modeled and observed behavior: A compromise between computation complexity and quality. In *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany*, pages 94–109, 2017.

[78] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and*

Analytics, Oct 2017.

[79] Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. Tuning alignment computation: An experimental evaluation. In *ATAED@Petri Nets/ACSD*, volume 1847 of *CEUR Workshop Proceedings*, pages 6–20. CEUR-WS.org, 2017.

[80] Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. Computing alignments of event data and process models. *T. Petri Nets and Other Models of Concurrency*, 13:1–26, 2018.

[81] Sebastiaan J. van Zelst, Boudewijn F. van Dongen, Wil M. P. van der Aalst, and H. M. W. Verbeek. Discovering workflow nets using integer linear programming. *Computing*, 100(5):529–556, 2018.

[82] Seppe K. L. M. vanden Broucke, Jorge Munoz-Gama, Josep Carmona, Bart Baesens, and Jan Vanthienen. Event-Based Real-Time Decomposed Conformance Analysis. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzzocrea, and Timos K. Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, volume 8841 of *Lecture Notes in Computer Science*, pages 345–363. Springer, 2014.

[83] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. Determining process model precision and generalization with weighted artificial negative events. *IEEE Trans. Knowl. Data Eng.*, 26(8):1877–1889, 2014.

[84] H. M. W. Verbeek. Decomposed Replay using Hiding and Reduction. In L. Cabac, L. Kristensen, and H. Rölke, editors, *PNSE 2016 Workshop Proceedings*, pages 233–252, Torun, Poland, June 2016.

[85] H. M. W. Verbeek. Decomposed replay using hiding and reduction as abstraction. *T. Petri Nets and Other Models of Concurrency*, 12:166–186, 2017.

[86] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. ProM 6: The Process Mining Toolkit. In M. La Rosa, editor, *Proc. of BPM Demonstration Track 2010*, volume 615 of *CEUR Workshop Proceedings*, pages 34–39,

Hoboken, USA, September 2010. CEUR-WS.org.

[87] H. M. W. Verbeek and Wil M. P. van der Aalst. Decomposed Process Mining: The ILP Case. In Fabiana Fournier and Jan Mendling, editors, *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, volume 202 of *Lecture Notes in Business Information Processing*, pages 264–276. Springer, 2014.

[88] H. M. W. Verbeek and Wil M. P. van der Aalst. Merging Alignments for Decomposed Replay. In Fabrice Kordon and Daniel Moldt, editors, *PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, volume 9698 of *Lecture Notes in Computer Science*, pages 219–239. Springer, 2016.

[89] H. M. W. Verbeek, Wil M. P. van der Aalst, and Jorge Munoz-Gama. Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining. *The Computer Journal*, pages 1–26, 2017.

[90] Jianmin Wang, Raymond K. Wong, Jianwei Ding, Qinlong Guo, and Lijie Wen. Efficient selection of process mining algorithms. *IEEE Trans. Services Computing*, 6(4):484–496, 2013.

[91] Lu Wang, YuYue Du, and Wei Liu. Aligning observed and modelled behaviour based on workflow decomposition. *Enterprise Information Systems*, 0(0):1–21, 2016.

[92] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011.

APPENDIX

A. DETAIL ON PARAMETER ESTIMATION OF HMMCONF

Here we detail various parts of the proposed approach in Chapter 7.

A.1. Forward probability (prior to observation update)

Let $P(X_{1:t-1}, Z_t)$ be the desired log forward probability prior to update from the current observation X_t at time t. We show that it can be computed using the conformance dependent state-transition probability, i.e., $P(Z_t|Z_{t-1}, X_{t-1})$, and the log forward probability from the previous time step, i.e., $P(X_{1:t-1}, Z_{t-1})$, as

$$P(X_{1:t-1}, Z_t) = \sum_{z \in Z} P(X_{1:t-1}, Z_{t-1} = z) P(Z_t|Z_{t-1} = z, X_{t-1}) \tag{A.1}$$

$$\begin{aligned}
P(X_{1:t-1}, Z_t) &= \sum_{z \in Z} P(X_{1:t-1}, Z_t, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_{1:t-2}, X_{t-1}, Z_t, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(Z_t, X_{1:t-2}|X_{t-1}, Z_{t-1} = z) P(X_{t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} \big[P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-2}|X_{t-1}, Z_{t-1} = z)\big] P(X_{t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-2}, X_{t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-1}, Z_{t-1} = z)
\end{aligned}$$

A.2. Forward probability

Let $P(X_{1:t}, Z_t)$ be the desired log forward probability with update from the current observation X_t at time t. We show that it can be computed using the conformance dependent state-transition probability, i.e., $P(Z_t|Z_{t-1}, X_{t-1})$, the conformance dependent observation probability, i.e., $P(X_t|Z_t)$, and the log forward probability from the previous time step, i.e., $P(X_{1:t-1}, Z_{t-1})$, as

$$P(X_{1:t}, Z_t) = P(X_t|Z_t) \sum_{z \in Z} P(X_{1:t-1}, Z_{t-1} = z) P(Z_t|Z_{t-1} = z, X_{t-1}) \tag{A.2}$$

$$\begin{aligned}
P(X_{1:t-1}, Z_t) &= \sum_{z \in Z} P(X_{1:t}, Z_t, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_{1:t-2}, X_{t-1}, X_t, Z_t, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_{1:t-2}, X_t, Z_t|X_{t-1}, Z_{t-1} = z) P(X_{t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_t, Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-2}|X_{t-1}, Z_{t-1} = z) P(X_{t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_t, Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_t|Z_t, X_{t-1}, Z_{t-1} = z) P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-1}, Z_{t-1} = z) \\
&= \sum_{z \in Z} P(X_t|Z_t) P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-1}, Z_{t-1} = z) \\
&= P(X_t|Z_t) \sum_{z \in Z} P(Z_t|X_{t-1}, Z_{t-1} = z) P(X_{1:t-1}, Z_{t-1} = z)
\end{aligned}$$

A.3. State-transition probability matrix

Here we detail the closed form update of the non-conforming state-transition probability matrices. As recalled from Section 7.3, the goal is to find parameters that maximizes

the expected log likelihood with respect to all latent state sequences with a constraint on their sum.

$$\begin{aligned} \underset{W^d_{a,i,j}}{\text{maximize}} \quad & \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=2}^{T} \log w_{z_{t-1},z_t}(x_{t-1}) \right] \\ \text{subject to} \quad & \sum_{j=1}^{|Z|} W^d_{a,i,j} = 1 \end{aligned}$$

While the matrix is substochastic where a case does not transition to another state once the final state is reached, deviating behavior might actually transition a case from the final state. We construct the Lagrangian:

$$\begin{aligned} \mathcal{L}(W^d_{a,i,j}) = \sum_{z_{1:T}} Q(z_{1:T}) \Bigg[& \sum_{t=2}^{T} \sum_{i=1}^{|Z|} \sum_{j=1}^{|Z|} \sum_{a=1}^{|A|} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \\ & \log w_{z_{t-1},z_t}(x_{t-1}) \Bigg] + \sum_{a=1}^{|A|} \sum_{i=1}^{|Z|} \lambda_{a,i} (1 - \sum_{j=1}^{|Z|} W^d_{a,i,j}) \end{aligned}$$

Similar to before, taking the partial derivatives and setting them to zero with respect to $W^d_{a,i,j}$ and $\lambda_{a,i}$ yield the parameter estimation:

$$\begin{aligned} & \nabla_{W^d_{a,i,j}} \mathcal{L}(W^d_{a,i,j}) \\ & = \nabla_{W^d_{a,i,j}} \sum_{z_{1:T}} Q(z_{1:T}) \Bigg\{ \sum_{t=2}^{T} \sum_{i=1}^{|Z|} \sum_{j=1}^{|Z|} \sum_{a=1}^{|A|} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \\ & \quad \log \Big[\textit{conf}(\vec{1}_i, a) W_{a,i,j} + (1 - \textit{conf}(\vec{1}_i, a)) W^d_{a,i,j} \Big] \Bigg\} + \sum_{a=1}^{|A|} \sum_{i=1}^{|Z|} \lambda_{a,i} (1 - \sum_{j=1}^{|Z|} W^d_{a,i,j}) \\ & = \sum_{z_{1:T}} Q(z_{1:T}) \Bigg[\sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \\ & \quad \frac{1 - \textit{conf}(\vec{1}_i, a)}{\textit{conf}(\vec{1}_i, a) W_{a,i,j} + (1 - \textit{conf}(\vec{1}_i, a)) W^d_{a,i,j}} \Bigg] - \lambda_{a,i} = 0 \end{aligned}$$

We only need to update $W^d_{a,i,j}$ when $\mathit{conf}(\vec{1}_i, a) = 0$ (note that an activity is either conforming or non-conforming to a given state), in which case:

$$= \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \frac{1}{W^d_{a,i,j}} \right] - \lambda_{a,i} = 0$$

$$W^d_{a,i,j} = \frac{1}{\lambda_{a,i}} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \tag{A.3}$$

The partial derivative of the Lagrangian can then be computed as:

$$\nabla_{\lambda_{a,i}} \mathcal{L}(W^d_{a,i,j}) = 1 - \sum_{j=1}^{|Z|} W^d_{a,i,j} = 0$$

$$= 1 - \sum_{j=1}^{|Z|} \frac{1}{\lambda_{a,i}} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} =$$

Then, the Lagrange multiplier can be derived so that it can be substituted into the update of the parameter.

$$\lambda_{a,i} = \sum_{j=1}^{|Z|} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}$$

$$= \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\} \tag{A.4}$$

$$W^d_{a,i,j} = \frac{\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}}{\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge x_{t-1} = a \wedge \mathit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}} \tag{A.5}$$

Similar to before, we can use the forward and backward probabilities to efficiently compute $\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\}$:

$$\begin{aligned}
&\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\} \\
&= \sum_{z_{1:T}} P(Z_{1:T} = z_{1:T} \,|\, X_{1:T} = x_{1:T}) \sum_{t=2}^{T} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\} \\
&= \sum_{t=2}^{T} \sum_{z_{1:T}} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\} P(Z_{1:T} = z_{1:T} \,|\, X_{1:T} = x_{1:T}) \\
&= \sum_{t=2}^{T} \sum_{z_{1:T}} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\} \frac{P(Z_{1:T} = z_{1:T}, X_{1:T} = x_{1:T})}{P(X_{1:T} = x_{1:T})} \\
&= \frac{1}{P(X_{1:T} = x_{1:T})} \sum_{t=2}^{T} \sum_{z_{1:T}} 1\{z_{t-1} = i \wedge z_t = j \wedge x_{t-1} = a\} P(Z_{1:T} = z_{1:T}, X_{1:T} = x_{1:T}) \\
&= \frac{1}{P(X_{1:T} = x_{1:T})} \sum_{t=2}^{T} \alpha_i(t-1) w_{i,j}(a) v_j(x_t) \beta_j(t)
\end{aligned}$$

This means that to update the weights for the non-conforming transition matrices:

$$W^d_{a,i,j} = \frac{\sum_{t=2}^{T} \alpha_i(t-1) w_{i,j}(a) v_j(x_t) \beta_j(t)\; 1\{x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}}{\sum_{j=1}^{|Z|} \sum_{t=2}^{T} \alpha_i(t-1) w_{i,j}(a) v_j(x_t) \beta_j(t)\; 1\{x_{t-1} = a \wedge \textit{conf}(\vec{z_{t-1}}, x_{t-1}) < 1\}} \tag{A.6}$$

A.4. Emission probability matrix

Same as for the state-transition probability matrix, the goal is to find parameters that maximizes the expected log likelihood with respect to all latent state sequences with a constraint on their sum.

$$\begin{aligned}
\underset{V^d_{k,j}}{\text{maximize}} \quad & \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=1}^{T} \log v_{z_t}(x_t) \right] \\
\text{subject to} \quad & \sum_{a=1}^{|A|} V^d_{a,j} = 1
\end{aligned}$$

Similar to the state-transition matrix, for conforming behavior, the matrix is substochastic where a case at the final state would not emit any activity observation. However, deviating behavior might mean that a case can emit an activity observation despite being in the final state. We construct the Lagrangian:

$$\mathcal{L}(V^d_{a,j}) = \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=1}^{T} \sum_{j=1}^{|Z|} \sum_{a=1}^{|A|} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1)\} \log v_{z_t}(x_t) \right] + \lambda_j (1 - \sum_{a=1}^{|A|} V^d_{a,j})$$

Taking the partial derivatives and setting them to zero with respect to $V^d_{j,a}$ and $\lambda_{j,a}$ yield the parameter estimation:

$$\nabla_{V^d_{a,j}} \mathcal{L}(V^d_{a,j}) = \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} \frac{1 - \textit{conf}(\vec{1_j}, a)}{\textit{conf}(\vec{1_j}, a) V_{a,j} + (1 - \textit{conf}(\vec{1_j}, a)) V^d_{a,j}} \right] - \lambda_j = 0$$

The non-conforming observation matrices $V^d_{a,j}$ would be updated when $\textit{conf}(\vec{1_j}, a) = 0$,

$$= \sum_{z_{1:T}} Q(z_{1:T}) \left[\sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} \frac{1}{V^d_{a,j}} \right] - \lambda_j = 0$$

$$V^d_{a,j} = \frac{1}{\lambda_j} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} \tag{A.7}$$

$$\nabla_{\lambda_j} \mathcal{L}(V^d_{a,j}) = 1 - \sum_{a=1}^{|A|} V^d_{a,j} = 0$$

$$= 1 - \sum_{a=1}^{|A|} \frac{1}{\lambda_j} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} = 0$$

$$\lambda_j = \sum_{a=1}^{|A|} \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\}$$

$$= \sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge \textit{conf}(\vec{z_t}, x_t) < 1\} \tag{A.8}$$

$$V^d_{a,j} = \frac{\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z_t}, x_t) < 1\}}{\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge \textit{conf}(\vec{z_t}, x_t) < 1\}} \tag{A.9}$$

Similar to before, we can use the forward and backward probabilities to efficiently compute $\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\}$:

$$\begin{aligned}
&\sum_{z_{1:T}} Q(z_{1:T}) \sum_{t=1}^{T} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} \\
&= \sum_{t=1}^{T} \sum_{z_{1:T}} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} Q(z_{1:T}) \\
&= \sum_{t=1}^{T} \sum_{z_{1:T}} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} P(Z_{1:T} = z_{1:T} \,|\, X_{1:T} = x_{1:T}) \\
&= \frac{1}{P(X_{1:T} = x_{1:T})} \sum_{t=1}^{T} \sum_{z_{1:T}} 1\{z_t = j \wedge x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} P(Z_{1:T} = z_{1:T}, X_{1:T} = x_{1:T}) \\
&= \frac{1}{P(X_{1:T} = x_{1:T})} \sum_{t=1}^{T} 1\{x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} \alpha_j(t) \beta_j(t)
\end{aligned}$$

This means that

$$V_{a,j}^{d} = \frac{\sum_{t=1}^{T} 1\{x_t = a \wedge \textit{conf}(\vec{z}_t, x_t) < 1\} \alpha_j(t) \beta_j(t)}{\sum_{t=1}^{T} 1\{\textit{conf}(\vec{z}_t, x_t) < 1\} \alpha_j(t) \beta_j(t)} \tag{A.10}$$