Partitionnement d'instances de processus basé sur les techniques de conformité de modèles

Mathilde Boltenhagen

université
PARIS-SACLAY

Process Instance Clustering Based on Conformance Checking Artefacts

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, Inria
Référent: : ENS Paris-Saclay

Thèse présentée et soutenue à Gif-sur-Yvette, le 21 octobre 2021, par

Mathilde BOLTENHAGEN

Composition du jury:

Pascal Poizat Professor, Université Paris Nanterre, LIP6	Président
Jochen De Weerdt Associate Professor, KU Leuven	Rapporteur
Marlon Dumas Professor, University of Tartu	Rapporteur
Fatiha Zaïdi Associate Professor, Université Paris Saclay	Examinateur
Marco Montali Professor, Free University of Bozen-Bolzano	Examinateur
Thomas Chatain Associate Professor, ENS Paris-Saclay	Directeur
Josep Carmona Associate Professor, Universitat Politècnica de Catalunya	Codirecteur

Thèse de doctorat

NNT: 2021UPASG060

“

La chance, tu la provoques. [...] Tu peux choisir de marcher au bord de la falaise ou tu peux choisir de marcher loin de la falaise. Moi, j'ai décidé que je veux vivre sur la falaise car j'ai une vue extraordinaire [...] Est-ce que j'ai peur de mourir ? Non. J'ai peur plutôt de ne pas vivre.

”

Mike Horn

Abstract

As event data becomes an ubiquitous source of information, data science techniques represent an unprecedented opportunity to analyze and react to the processes that generate this data. *Process Mining* is an emerging field that bridges the gap between traditional data analysis techniques, like Data Mining, and Business Process Management. One core value of Process Mining is the discovery of formal process models like Petri nets or BPMN models which attempt to make sense of the events recorded in logs. Due to the complexity of event data, automated process discovery algorithms tend to create dense process models which are hard to interpret by humans. Fortunately, *Conformance Checking*, a sub-field of Process Mining, enables relating observed and modeled behavior, so that humans can map these two pieces of process information. Conformance checking is possible through alignment artefacts which associate process models and event logs. Different types of alignment artefacts exist, namely alignments, multi-alignments and anti-alignments.

Currently, only alignment artefacts are deeply addressed in the literature. It allows to relate the process model to a given process instance. However, because many behaviors exist in logs, identifying an alignment per process instance hinders the readability of the log-to-model relationships.

The present thesis proposes to exploit the conformance checking artefacts for clustering the process executions recorded in event logs, thereby extracting a restrictive number of modeled representatives. *Data clustering* is a common method for extracting information from dense and complex data. By grouping objects by similarities into clusters, data clustering enables to mine simpler datasets which embrace the similarities and the differences contained in data. Using the conformance checking artefacts in a clustering approach allows to consider a reliable process model as a baseline for grouping the process instances. Hence, the discovered clusters are associated with modeled artefacts, that we call *model-based trace variants*, which provides opportune log-to-model explanations.

From this motivation, we have elaborated a set of methods for computing conformance checking artefacts. The first contribution is the computation of a unique modeled behavior that represents of a set of process instances, namely multi-alignment. Then, we propose several alignment-based clustering approaches which provide clusters of process instances associated to a modeled artefact. Finally, we highlight the interest of anti-alignment for extracting deviations of process models with respect to the log. This latter artefact enables to estimate model precision, and we show its impact in model-based clustering. We provide SAT encoding for all the proposed techniques. Heuristic algorithms are then added to deal with computing capacity of today's computers, at the expense of loosing optimality.

To the memory of my late grand-father, Pépé, René Guillon,
(Sept.1927- Sept.2021)

Acknowledgement

Les grand-parents renferment un coffre fort de connaissances et de souvenirs qui, en étant transmis aux nouvelles générations, font mûrir les plus jeunes, leur donnant l'ambition de continuer à valoriser les trésors offerts par la vie. Le fruit de mon travail de recherche est, sans aucun doute, l'effet papillon de leur travail et persévérance. Je les remercie du fond du coeur.

This thesis would not have been possible without the supervision of Thomas Chatain and Josep Carmona. I would like to express my deepest thanks for their extraordinary presence, advice and kindness. I am very grateful for the three years of work we spent together. I will remember their management and scientific skills that allowed us to obtain a set of papers done in a very short time. I will also remember the good times we shared.

I thank Jochen De Weerdt and Marlon Dumas for their careful reviews and encouraging reports. I thank Fatiha Zaïdi, Pascal Poizat and Marco Montali for the honour they give me by participating in my jury. Moreover, I would like to thank the Process Mining community for its warm welcome.

I thank the colleagues of the lab and especially Igor for his meaningful care during the three years. I thank Philippe for the enriching coffee breaks, and Hugues, Patricia and Marie-France for their efficient administrative support. I also would like to thank Corbin and Da-Jung for the time spent together.

This PhD contract has existed thanks to Marc Plantevit that trusted me and shared my profile to his network. I would like to thank him too.

En outre, j'ai la chance d'être entourée de personnes extra-ordinaires, à savoir, ma famille, mes amis, mes amours. Je les remercie tout d'abord d'être les merveilleuses personnes qu'ils sont.

Certains ont joué un rôle majeur dans l'accomplissement de ma thèse. Je remercie d'abord Mickaël avec qui j'ai pleinement partagé cette expérience d'étudiant-chercheur. De même, je remercie Benjamin et Laurine pour les multiples discussions scientifiques et philosophiques. Ce fût une joie immense d'avoir abouti un papier scientifique avec des amis dans le cadre de ma thèse. Je les remercie pour leur implication et intérêt.

A l'image d'une thèse ordinaire, mes trois années de doctorat ont été parsemées de joie, doute et déception, émotions apaisées par l'écoute attentive de ma complice Laurine. Je la remercie infiniment d'avoir été plus que jamais présente à mes côtés sans quoi cette

thèse n'aurait vu le jour. Je la remercie aussi pour tous nos fous rires à l'autre bout de la France, nos discussions littéralement sans fin, nos projets ambitieux, et tous les merveilleux moments passés ensemble. Je me souviendrai longtemps de cette école de recherche à Riga.

Je remercie ma mère pour l'éducation exemplaire, alliant santé, simplicité et sociabilité, qu'elle a donné à mon frère et moi. Une éducation qui nous a permis de nous épanouir et de nous projeter avec ambition.

Je remercie mon frère, Elie, avec qui je peux débattre et rire des projets les plus fous comme ce doctorat.

Je remercie Patrice d'être entré dans nos vies récemment. Je le remercie pour son intérêt à passer du temps avec Elie et moi. Je n'oublie pas sa patience à m'écouter répéter des présentations techniques pendant la préparation du repas, et ce, en anglais. Je le remercie aussi pour son adaptation si naturelle à nos côtés. Qu'il soit assuré de ma gratitude de sa présence parmi nous.

Je n'oublie évidemment pas mon compagnon de vie, Arthur, qui m'a soutenue, supportée, longuement écoutée et encouragée, durant cette période de thèse, mais aussi de télétravail. Je le remercie particulièrement pour sa patience dans les moments durs, ses discours réconfortants et nos tranches de rigolade. Qu'il soit assuré de la gratitude que j'éprouve de l'avoir à mes côtés quotidiennement.

Dans mes amis de longue date, je remercie Aline et Marie, qui, malgré les années, m'apportent toujours autant de soutien et d'énergies positives. Je remercie bien sûr Romane pour tous les moments exceptionnels que nous avons vécus ensemble qui m'ont fait voir grand et loin. J'ai tant hâte de la revoir. Je remercie Guillaume pour sa joie de vivre et ses décisions exemplaires, Igor pour ses reflexions profondes, Olivier pour son recul exagéré, Théo pour sa bonne humeur toujours présente, Pauline pour sa douceur et son sens de la discrétion, Lubin pour son humour décalé et Maïwenn pour sa gratitude infinie. Des qualités admirables qui me stimulent lorsque nous nous voyons.

Je remercie aussi, et encore, Manu, Lucien, Benjamin, Laurine, Thomas, Brice, Carlos, Mehdi, Robin, Vincent, Arnaud, Antoine, Mickaël, Théophile, Thomas, Thomas, Axel, Nathan, Titouan, Tangui, Loïc, Charles et Naama sans qui mes années d'études et n'auraient pas été les mêmes. Plus particulièrement, je remercie Manu pour sa spontanéité délirante, Benjamin pour son impressionnante ambition, Charles pour son intérêt sans limite, Loïc et Tangui pour leur côté fêtard, Naama pour son ouverture d'esprit exemplaire et Thomas pour son caractère décisif. Tous me rappelant la complexité intrigante de notre monde. Je remercie aussi Clara pour sa motivation à créer et son émerveillement si encourageant à chaque projet. Enfin, je remercie Michaël et Cédric qui ont éveillé en moi une passion, l'informatique.

Merci à tous, merci la vie.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Chapter Overview

This chapter gives an introduction to this doctoral thesis. The first section presents the general context of today's event data situation. The following section defines Process Mining. We zoom on conformance checking and log trace clustering which draws the context of the thesis. In Section. 1.3 we present the motivation of the research and, finally, Section 1.4.1 presents the content of the thesis from the research challenges to the contributions and tools.

1.1 Data Analysis of Business Processes

Nowadays, digitalization has become an indispensable part of everyday life of people and companies such that the amount of produced, captured, copied, and consumed data would have reached 59 zettabytes by the end of 2020 according to the International Data Corporation (IDC) [1]. The forecast for year 2025 raises to 165 zettabytes and has been accelerated by the COVID-19 pandemic which contributes in data creation and consumption.

The interest in data analysis rapidly grew in organizations as the quality of data impacts decision making and business strategy [66]. In 2016, a IBM study reports that poor data quality leads a US cost of $3 trillion per year [98]. A real wish to mitigate this loss is illustrated by the number of data scientist positions on the top emerging jobs since several years [2]. Data analysis plays an important role in various organizations from public health care to industries and even for pandemic crisis in order to assess, optimize and warrant decision making [106].

In this thesis, we tackle event data recorded in logs. Structured as a collection of events, those data describe the processes that appear in organizations. The particularity of those data is the scoped dimensions. Fig. 7.1 presents an artificial example of event data. Every line represents an event. Classical data mining techniques assess those data by working on

Case Identifier	Timestamp	Activity	Resource 1	Resource 2
239U	16-03-2020:11.02	open	Proc.1	Silent
187V	16-03-2020:11.07	open	Proc.2	Silent
187V	16-03-2020:11.09	write	Proc.2	Silent
239U	16-03-2020:11.31	read	Proc.1	Silent
239U	16-03-2020:12.20	write	Proc.2	Silent
239U	16-03-2020:12.20	close	Proc.2	Console
...	...	...	...	...
982R	17-03-2021:12.20	close	Proc.2	Console

Figure 1.1: Example of Event Data

Case Identifier	Sequence of Activities
239U	$\langle open, read, write, close \rangle$
187V	$\langle open, write, close \rangle$
...	...
982R	$\langle open, write, close \rangle$

Figure 1.2: Schematic Sequential View of Event Data

features which are usually the dimensions, i.e., the columns, of the dataset. Then, events can be grouped, predicted and more generally mined. However the order of the activities is a key information in process analysis and this aspect is not given by the events themself. By grouping events by case identifier and ordering them by timestamp we obtain the behavioral information as schematized in Fig. 7.2

Example 1.1.1 (Event Data)**.** *The first event of Fig. 7.1 belongs to case* $239U$*. It describes the activity "open" and is associated to resource 1 of type "Proc.*1*" and resource 2 set as "silent". In fact, the case* $239U$ *contains several events. We can concat its sequence of activities:* $\langle open, read, write, close \rangle$ *as shown in Fig. 7.2.*

Technical Details

Event logs are commonly stored in XES files a XML type of files which allows to group the events according the case ordered by timestamp, thus by keeping all the resource attributes. An example is given in Fig. 1.3.

Then, from the sequences of events, the aim is to discover *causality*, *concurrency*, *choice* and *loop* behavior. A causal dependency is simply the pattern of having an ordering between activities. Concurrency draws the notion of parallelism between activities, where a group of activities can appear in different orders. Choices allow to differentiate several possible behaviors. Finally, loops is the repetition of operations in processes. Discovering these patterns apace becomes a complex task due to the amount of different behaviors

```
<?xml version="1.0" encoding="UTF-8" ?>
<log>
        <string key="concept:name" value="logjam"/>
        <trace>
                <string key="id" value="982R"/>
                <event>
                        <string key="concept:name" value="open"/>
                        <date key="time:timestamp" value="2020-16-03T15:07:00"/>
                        <string key="org:resource1" value="Proc.1"/>
                        <string key="org:resource2" value="Silent"/>
                </event>
                <event>
                        <string key="concept:name" value="write"/>
                        <date key="time:timestamp" value="2020-16-03T23:34:00"/>
                        <string key="org:resource1" value="Proc.1"/>
                        <string key="org:resource2" value="Silent"/>
                </event>
                <event>
                        <string key="concept:name" value="close"/>
                        <date key="time:timestamp" value="2021-17-03T12:20:00"/>
                        <string key="org:resource1" value="Proc.2"/>
                        <string key="org:resource2" value="Console"/>
                </event>
        </trace>
</log>
```

Figure 1.3: XES Format

contained in logs [76].

In addition, event data bring three fundamental difficulties: the *lack of negative information*, the *presence of history-dependent behavior* and the *presence of noise*, i.e., errors, in logs [139]. Indeed, recorded data contained in log are only positive information that we want to learn. There is no false recorded behavior that would help to understand the limitations of system. Moreover, business processes are inscribed in time and describe a specific moment which makes it hard to generalize. Last but not least, quality of the recorded logs is an important barrier in log analysis because one has to determine if some events are missing or wrong before proceeding with process analysis.

In the literature, some fields like Sequences Pattern Mining focus on particular behaviors in sequential data [59]. Similarly, Deep Learning approaches have shown huge improvements in sequence analysis like in Natural Language processing, where sentences are sequences of words [94]. The context and order of items in sequences are of great interest in research. However the aforementioned fields do not provide start-to-end understandable models which is core source of *explainability* required in *Business Process Management* where decision makings must be reliable. Process models proved to be a key element for describing, analyzing, monitoring and optimizing the execution of the processes [100]. Thanks to Process Mining that bridges the gap between Business Process Management and Data Science by providing a bunch of techniques to discover and verify

process models [124].

1.2 Process Mining

Process Mining is a recent field that emerged in the last decades to extract information from event logs by producing process models-centric analytics. Fig. 1.4 depicts the matters of the field [119].

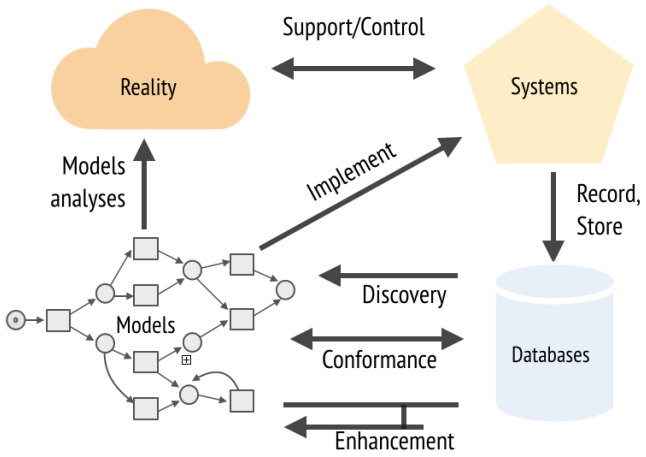


Figure 1.4: Positioning of the Three Main Types of Process Mining: Process Discovery, Conformance Checking, and Enhancement

Real process instances are supported by systems which collect the event data. Thanks to *Process Discovery* algorithms, data analysts can mine process models representing the operations that structure their organizations [118]. These automated process discovery methods take as input an event log, and produce a business process model that captures the control-flow relations between tasks that are observed in the log. The produced models are usually Petri nets or BPMN (Business Process Management Notation) models because they allow to formally describe causality, concurrency, choice and loop behavior [9]. We formally present Petri net models in the next chapter.

As business decisions rely on these discovered models, it is crucial to ensure the conformance of them with respect to the recorded process executions. This model-to-log comparison is known as *Conformance Checking* [25]. Finally, *Enhancement* is the ability to modify and correct the models to better assess the behaviors contained in organizations.

1.2.1 Conformance Checking

In this thesis we zoom on Conformance Checking methods and assume an existing process model representing a corresponding event log. This assumption is realistic as a set of

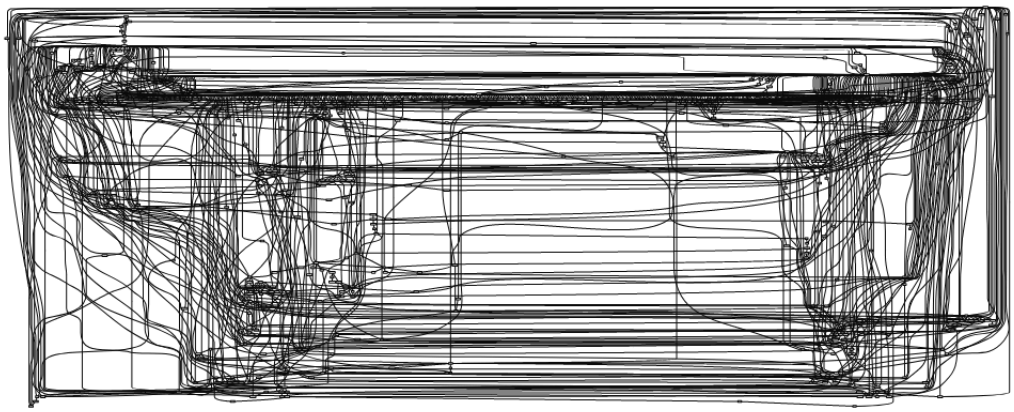


Figure 1.5: Spaghetti Process Describing the Diagnosis and Treatment of 2765 Patients in a Dutch Hospital, Example from [120]

algorithms exist to discover a process model from a log [121, 140, 125, 75, 10, 81]. Precisely, a recent review by Augusto et al. identifies 35 groups of discovery algorithms in the literature [9]. Furthermore, in many contexts, e.g. in Process-Aware Information Systems, process models are often available [49].

Once a process model is discovered or given, the matter is to check the relevance of it with respect to the real behaviors. Conformance checking aims at relating modeled and observed behavior. The main objective is to measure the differences between the obtained modelization and the recorded processes. As of today, there are four quality criteria for good process models:

- **Fitness (a.k.a. recall):** indicates how much of the observed behavior in data is described by the process model.
- **Precision:** measures how much modeled behavior exists in the event log.
- **Generalization:** represents the ability of a model to correctly capturing parts of the system that have not been recorded
- **Simplicity:** measures the understandability of a process model by limiting the size and complex structures of the resulting model.

A trade-off between the quality criteria is one big dilemma of the field because of the high complexity of the involved data and the corresponding produced models [120]. For instance, a known analogy of the problem is the spaghetti-like models. Fig. 1.5 show an example given by [120] where the diagnosis and treatment of 2765 patients in a Dutch hospital are described. Despite representing all or most of the behaviors of the log, this kind of product is so dense that it is difficult to comprehend. Moreover, those models usually contain many behaviors that are not present in the log as a result of the many possible paths that they design. We say that the model is fitting, i.e., its fitness is high

since the recorded processes are represented by the model, but imprecise because it models many unobserved behaviors, and has a low simplicity as the structure is very dense. This example triggers the challenge of finding a good compromise between the conformance checking criteria.

Moreover, the metrics associated to the criteria are still discussed by the process mining community [109]. The fitness metric is the only unanimously accepted one. It used *alignments*, a conformance checking artefact that relates a log sequence to a run of the corresponding process model [6]. The key interest of such artefacts is the *explainability* of the results.

To deal with the conformance criteria, some studies propose to reduce the problem to local parts of the processes [83, 110]. Instead of getting a global model of the all the recorded operations, the approach aims at learning local process models representing subprocesses contained in logs. Then, the produced process models are less complex and the trade-off between the conformance checking criteria is more achievable.

Another method to reduce the complexity of event data and obtain more accurate models is to analyze subsets of log instances separately.

1.2.2 Trace Variants and Process Instance Clustering

In Process Mining, the notion of *trace variants* refers to the unique behaviors in logs omitting the resource attributes [114]. They are extracted to explore the different processes that appear in organizations. However, in practice, the number of different process instances is large and not interpretable by decision makers.

Example 1.2.1 (Trace Variants)**.** *Fig. 7.2 shows the sequences of activities. We observe that cases* $187V$ *and* $982R$ *execute the same behavior. When extracting the variants, only one version of this sequence of activities is kept.*

In another hand, a main task for grouping objects in Exploratory Data Analysis is *data clustering*. Data clustering is the task of partitioning objects in different groups known as clusters, in which the objects are similar. Process instance clustering is then the partition of log instances in sublogs such that the clusters group similar processes. This topic of research has shown a large interest in process mining in the two last decades with 103 relevant works [144]. Thus, the similarity of process instances has been approached from several perspectives:

- On the first hand, the study of the **control-flow** given by the log sequences allows grouping process instances according to the behavior they describe. In other words, the activities that appear in the system are assessed. These clustering methods range from the study of the frequency of the activities [108] to the study of patterns [63, 39, 21, 78].

- On the other hand, **context perspective** approaches provide clustering based on the data attributes like "Resource 1" and "Resource 2" of Fig. 7.1. These techniques get closer to classical data mining [133].
- Some works deal with the two approaches [108, 79].

The outputs of those works show a real interest of process instance clustering in process discovery. Instead of learning a model representing the entire log, the idea is to mine a process model per cluster. Then, the produced models give a better compromise between the quality criteria thanks to the homogeneity of the clusters.

A perspective missing of the last few paragraphs is the existence of a process model. There, trace variants and clusters of process instances are learned and extracted from the event log only.

1.3 Research Motivation

Once a process model has been validated by its process owner, the practitioner can benefit from the knowledge of this model by using it as a baseline for log analysis. Hence, trace variant extraction and process instance clustering can use this reliable process model as input. This idea is in contrast to the aforementioned situation where the motivation is to learn simpler models from sublogs. Here, the process model can be complex and the objective is to extract simpler artefacts from it. This perspective is motivated by the complexity of the process models produced by the discovery algorithms that mainly prioritize fitness [120]. Since the learned model contains the behavioral information and a visualization of it which known by the process owner, a log analysis based on it gives a novel view for decision making.

In this thesis, we propose to fill this gap and present approaches that use conformance checking techniques to represent sublogs based on a reliable process model. Thus, we allow partitioning event log and extract modeled artefacts that we use as *model-based trace variants*.

1.4 Thesis Overview

This thesis gives definitions, algorithms and applications of conformance checking artefacts for finding good model-based trace variants, i.e., process instance representatives based on a reliable process model, through clustering approach. In this section, we outline the content of this thesis that reveals the main contributions. Then, we present the research challenges and methodology that we have identified to develop our research. In Section. 1.4.3 we provide a summary of all the research pieces that have been previously published in international peer-reviewed workshops, conferences and journals. All contributions are supported by implementation and experiments. Thus, we conclude the introductory chapter with a presentation of the tools.

1.4.1 Thesis Structure

The present thesis contains 6 chapters including the current one. Chapter 2 gives the necessary background. Then Chapters 3 to 5 give the main contributions and Chapter 6 concludes the thesis. For each contribution of the main chapters, illustrated in Fig.1.6, Fig.1.7 and Fig.1.8, we developed an optimal algorithm such that desired outputs can be computed for demonstration. Those algorithms allow us to present proofs of concept of the methods. Then, we propose heuristics to handle large inputs as they appear in industry. We experimented every work with both artificial and real-life data.

The first contribution, schematized in Fig.1.6, is the development of two algorithms for computing *multi-alignments*. Multi-alignment is a conformance checking artefact that relates many log sequences to a unique modeled sequence. This artefact can help one to get an overview of a log or a sub-log and then, stands as model-based trace variant. The proposed algorithms for computing multi-alignments extend to classical alignments. Consequently, this chapter provides an novel optimal encoding and several heuristics for computing both alignments and multi-alignments.

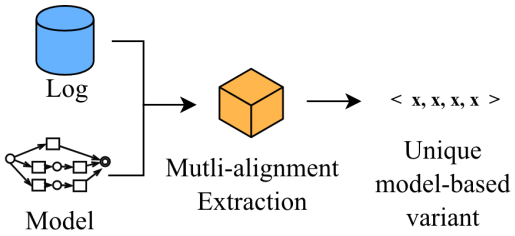


Figure 1.6: Schematization of Chapter 3

The disadvantage of multi-alignment is that it is a single artefact that represents all the sequences given as input. Thus, multi-alignment extraction fits well when the log is homogeneous but becomes less appropriate when the log contains several types of behaviors. In the latter situation, one want to separate the behaviors in different groups such that the modeled variant is accurate to each group. Chapter 4 solves this problem by proposing a set of 3 clustering methods based on alignments. Then, from a model and a log, the algorithms partition the log sequences into clusters and provide a variant per cluster.

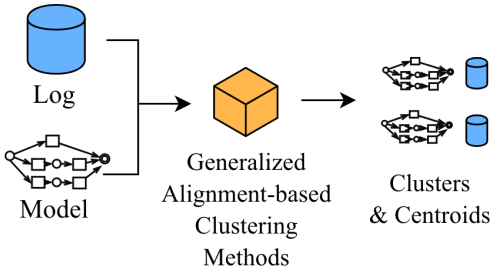


Figure 1.7: Schematization of Chapter 4

Both previous methods assume a process model and extract model-based trace variants of a set of log sequences based on this model. However, the quality of the input model makes varying the results of the methods. For this purpose, we present in Chapter 5 another conformance checking artefact entitled *anti-alignment* which aims at measuring precision of process models. As shown in Fig. 1.8, the algorithm takes a model and a log as input and extracts one of the most deviant modeled sequence with respect to the log. This latter can then be used to compute the precision of the model.

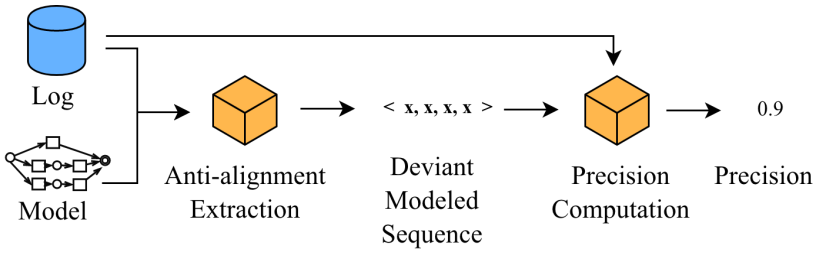


Figure 1.8: Schematization of Chapter 5

In a nutshell, this thesis focuses on three conformance checking artefacts in the context of log trace clustering. We now show how those works are positioned in the literature.

1.4.2 Research Challenges and Methodology

In 2012, Van der Aalst W. presented in [117] an overview of the challenges in Process Mining. As our motivation is to get model-based trace variants though clustering approach, we tackle Challenge 9 that highlights the interest in bridging the gap between process mining and other techniques like data mining and optimization:

Combining Process Mining With Other Types of Analysis: *The challenge is to combine automated process mining techniques with other analysis approaches (optimization techniques, data mining, simulation, visual analytics, etc.) to extract more insights from event data.*

Our proposed methods are based on conformance checking artefacts that rely on quality of process models. Moreover, we work on anti-alignment which is a conformance checking artefact used in a precision metric. Thus, the thesis also contributes to Challenge 6:

Balancing Between Quality Criteria such as Fitness, Simplicity, Precision, and Generalization: *There are four competing quality dimensions: (a) fitness, (b) simplicity, (c) precision, and (d) generalization. The challenge is to find models that score good in all four dimensions.*

Now, to define valuable methods for extracting model-based trace variants, we identify the following research goals:

G1 (Definitions): formally defining the model-based trace variants and the conformance checking artefacts

G2 (Settings): obtaining a restrictive number of model-based trace variants for human analysis such that differences and similitudes are well highlighted and interpretable for decision making, i.e., defining clustering parameters,

G3 (Proof of Concept): producing algorithms that get the model-based trace variants and developing proofs of concept through a set of experiments,

G4 (Analysis): checking the accuracy of the results in accordance to the process executions contained in logs and the state-of-the-art methods

These steps have been tackled in both the different contributions and this thesis.

1.4.3 List of Papers

Our contributions vary from theoretical definitions to algorithm optimization and experiments. They have been published in peer-review papers that we list below. For each paper, we give the tackled research goals and detail the propose piece of solution.

C1: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *Generalized Alignment-Based Trace Clustering of Process Behavior*, International Conference on Applications and Theory of Petri Nets and Concurrency (June 2019)

This paper tackles **G1** to **G3** despite that the notion of model-based trace variants was not yet set and the algorithm worked only for artificial logs. The paper presents different process instance clustering methods based on an existing process model. The content is formally defined and associated to a SAT encoding. We give an adaptation of the usual definitions of inter- and intra-cluster distance for Petri net and set a list of quality criteria for finding good clustering.

C2: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *Encoding Conformance Checking Artefacts in SAT*, Business Process Intelligence Workshop, in conjunction with BPM' 2019 (September 2019)

This paper brings a novel implementation of conformance checking artefacts which fits **G3** perfectly. Two of the three artefacts presented in this paper were not computable before this contribution.

C3: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *Optimized SAT encoding of conformance checking artefacts*, Computing, Springer-Verlag GmbH Austria, part of Springer Nature

This paper is the journal extension of the previous paper noted **C2**. It tackles the same goal but bring much faster and reliable implementation.

C4: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *Model-based trace variant analysis of event logs*, Information Systems, Elsevier B.V.

Similarly to the previous contribution, this paper is the journal extension of contribution **C1**. In this paper, the notion of model-based trace variants emerges. We give large experiments and a comparison to another method of the state-of-the-art which completes **G3** of **C1**.

C5: Thomas Chatain, Mathilde Boltenhagen, and Josep Carmona *Anti-alignments—Measuring the precision of process models and event logs*, Information Systems, Elsevier B.V.

This contribution is a journal paper that presents *anti-alignment* and its use for measuring precision of process model. In the context of this thesis, we will see how anti-alignment helps in reaching good clustering results, i.e., it tackles **G4**.

C6: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *A Discounted Cost Function for Fast Alignments of Business Processes* , Business Process Management Conference (September 2021)

The clustering methods presented in this thesis uses the conformance checking entitled alignment as a key element for grouping the process instances. Naturally, the efficiency to compute those artefacts influences the clustering efficiency. This is why, we decided to work on alignment heuristics. This contribution aims at penalizing prefix of the input sequences. This contribution tackles **G3**.

At the time of the thesis writing, a paper is still under review. We mark it with symbol *:

C7*: Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona *An A*-Algorithm for Computing Discounted Anti-Alignments in Process Mining*, under revision for ICPM'2021

Observing the previous contribution **C6***, we projected the idea to anti-alignment thus making the algorithm working for real-life logs which was not possible before in **C5**. Then we classify this contribution to the **G3**.

Finally, some papers (workshops) are out of the scope of the present thesis despite being in part of Process Mining contributions.

C8: Mohammadreza Fani Sani, Mathilde Boltenhagen, and Wil M.P. van der Aalst *Prototype Selection using Clustering and Conformance Metrics for Process Discovery*, Business Process Intelligence Workshop, in conjunction with BPM' 2020

This paper gives a preprocessing method for process model discovery. In general process models are very dense and complex which gives low precision. In this paper, we incrementally use some process instance variants, entitled *prototypes*, to learn a simpler model still accurate to the log. We use the F-measure to obtain a good balance between fitness and precision.

C9: Mathilde Boltenhagen, Benjamin Chetioui, and Laurine Huber An *Alignment Cost-Based Classification of Log Traces Using Machine-Learning*, First International Workshop on Leveraging Machine Learning in Process Mining, in conjunction with ICPM'2020

Not to confuse with data clustering, data classification is the task of predicting label of objects. In this paper, we present an overall idea of conformance checking classification in order to reduce the computation of alignments. By learning which log instances are observed in the process model, we aims at predicting, with a threshold, the binary class of them, i.e., are these behaviors modeled? As of today, this work is still in progress.

1.4.4 Tools

The approaches and methods defined in this thesis are all implemented in open source projects available on github [1]. In some case, two implementations are even available for the same work which reaches more the community.

DarkSider: DarkSider is the first prototype tool that implemented the optimal encoding conformance checking methods for model-based trace variants. Started by Thomas Chatain in 2016, the software is implemented in Ocaml, a functional language that allows a good typography useful for SAT encoding. The tool is a command line software which requires a terminal only. Contributions **C1** and **C2** are implemented in this tool.

da4py: due to the abundance of interest in Python in the Data Science community, da4py is the traduction of DarkSider in Python 3. Then, we hope to motivate the community to contribute in our work. Moreover, it is more convenient for users when a unique script is used, for instance a Jupiter notebook. In the implementation view, optimal encoding of the methods performs much better in Python than in Ocaml thanks to the pysat library that bridges the gap between Python and all the different outputs of the SAT solvers [68]. Contributions **C3**, **C4** and **C5** belong to this library.

pm4py: finally contributions **C6** and **C7*** are implemented in a copy of pm4py Python library.

[1] https://github.com/BoltMaud

Chapter 2

Background

Chapter Overview

This chapter provides the notations and background needed to understand the remainder of the thesis. In Section 2.1, we present the process mining structures. Then, in Section 2.2 the notion of alignment is detailed. Finally, Section 2.3 gives some formal concepts like SAT encoding and complexity.

2.1 Structures

Process Mining brings a set of different process models including, but not limited to, process trees, Petri nets, FlowCharts, and BPMN models. Similarly, event data can be exploited in several ways. We precise the formal data structures used in this thesis in the followings.

2.1.1 Event Logs

Event data, as presented in the introduction section, contain various meta-data. The present work focuses on the sequences of activities entitled *log traces*.

Definition 1 (Log Traces)**.** *Let Σ be a set of activities. We define a* log *L as a finite multiset of sequences $\sigma \in \Sigma^*$, which we refer to as* log traces*.*

A log trace σ corresponds to a behavior contained in the event log. It can be extracted by grouping events by identifier, ordered by timestamp and noted by the name of the activity of the events.

Technical Details

In XES files, the traces are already extracted in **trace** nodes. The timestamp are given in date nodes with key **time:timestamp** and activity names are of type string, tagged by a key entitled **concept:name**. An example is given in Fig. 1.3.

$$\begin{gathered}\langle open, read, write, close\rangle \\ \langle open, read, write, close\rangle \\ \langle open, read, write, close\rangle \\ \langle open, write, close\rangle \\ \langle open, wait, write, close\rangle \\ \langle open, write, read, wait, wait, close\rangle \\ \langle open, wait, write, read, close\rangle \\ \langle write, wait, wait, close\rangle\end{gathered}$$

Figure 2.1: Log L

Example 2.1.1 (Log trace)**.** *An example of log traces is given in Fig. 2.1. The first sequence* $\sigma = \langle open, read, write, close\rangle$ *correspond to the behavior of identifier* $239U$ *of Fig. 7.2.*

Classically, some log traces appear many times in the log: this simply means that they correspond to frequent behavior. For this reason, one usually groups equivalent log traces together and considers them as several instances of the same *trace variant* [114].

Definition 2 (Trace Variants)**.** *Given a log* L*, the* trace variants *are the unique sequences contained in* L*.*

Example 2.1.2 (Trace Variants)**.** *The trace variant* $\langle open, read, write, close\rangle$ *represents the three first log traces.*

In another hand, as presented in the introduction, one can use clustering methods that aim at partitioning data by similarity in order to allow slight differences in groups. We give a general definition of trace clustering.

Definition 3 (Trace Clustering)**.** *Given a log* L*, a trace clustering over* L *is a partition over a (possibly proper) subset of the traces in* L*.*

2.1.2 Process Models

We use labeled Petri nets as process models. Those models formally define causality, concurrency, choice and loop behaviors.

Definition 4 (Process Model (Labeled Petri Net System) [87])**.** *A Process Model defined by a* labeled Petri net system *(or simply* Petri net*) is a tuple* $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$*, where* P *is the set of places,* T *is the set of transitions (with* $P \cap T = \emptyset$*),* $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation,* m_0 *is the initial marking,* m_f *is the final marking,* Σ *is an alphabet of actions and* $\Lambda : T \to \Sigma \cup \{\tau\}$ *labels every transition by an action or as silent.*

Semantics. A *marking* is the set of places that contain tokens for a given instant. A transition x can *fire* if all the places before x, noted ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$, are marked. When a transition fires, all the tokens in ${}^\bullet x$ are removed and all the places in $x^\bullet \stackrel{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$ become marked. A marking m' is *reachable* from m if there is a sequence of firings $\langle t_1 \dots t_n \rangle$ that transforms m into m', noted $m[t_1 \dots t_n \rangle m'$. The set of reachable markings from m in N is denoted by $RS(N, m)$.

Labels of transitions are the activity names and the places correspond to state in the processes. Black transitions refer to silence transition meaning that no activity is represented. They are used for the consistence of Petri nets.

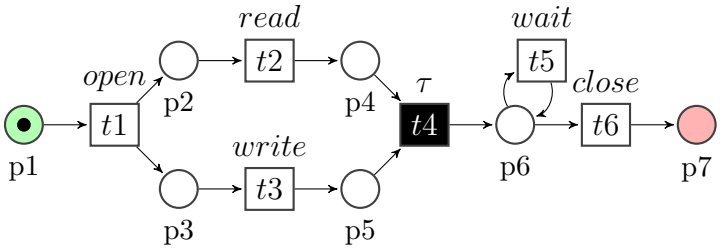


Figure 2.2: A Model N

Example 2.1.3 (Process model, marking, firing sequence)**.** *Fig. 7.3 shows a process model* N *of activities in* Σ *of log* L*. The initial marking is* $m_0 = \{p1 : 1\}$*. When firing transition* $t1$*, the token is removed from* $p1$ *and places* $p2$ *and* $p3$ *both receive a token.*
Transitions $t1$ *and* $t4$ *form a concurrency pattern where activities* ***read*** *and* ***write*** *can appear in parallel. The firing sequence* $\langle t1, t2, t3, t4 \rangle$ *represents the sequence of activities* $\langle open, read, write, \tau \rangle$ *and reaches marking* $m' = \{p6 : 1\}$*.*

Definition 5. *(Boundedness, Safeness) A Petri net* $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ *is* k-bounded *if no marking reachable from* m_0 *assigns more than* k *tokens to any place. A Petri net is* safe *if it is 1-bounded.*

In this thesis, we tackle process model in the class of sate nets.

Definition 6. *(Easy Soundness) Let* $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ *be a Petri net over activities in* Σ*.* N *is an* easy sound *Petri net if and only if* $m_f \in RS(N, m_0)$ *.*

Technical Details

In Process Mining, Petri nets are often reduced to **Workflow Petri nets** which are easy sound and safe Petri nets whom initial and final marking are restricted to one place each. As our methods work for a larger class of Petri nets, they can be applied to any workflow Petri net.

Modeled behaviors of a process model N are the full runs of N defined below.

Definition 7 (Full runs)**.** *A* full run *of a model N is a firing sequence $\langle t_1 \dots t_n \rangle$ of transitions that can transform the initial marking m_0 of N to the final marking m_f of N. We note $Runs(N)$ the full runs of N.*

Example 2.1.4 (Full runs)**.** *The run $\langle t1, t2, t3, t4, t6 \rangle$ is a full run of the model of Fig. 7.3.*

To alleviate reading, we write the modeled sequences directly with the labels of the transitions in the rest of the thesis.

2.2 Alignments

In the previous section, we presented an example of a log and a process model. But the log traces are not all represented by the model. For instance, the recorded sequence $\langle write, wait, wait, close \rangle$ is not described by the model. Then, to relate these traces to the model, Aray et al. presented alignments which is today the main conformance checking artefact [4].

2.2.1 Definition

In this thesis, we present the definition of alignments as runs of the process model.

Definition 8 (Alignment)**.** *Given a log trace σ of L and a model N, an (optimal)* alignment *of σ to N is a full run $u \in Runs(N)$ which minimizes the distance $dist(\sigma, u)$ between u and σ, where dist is a distance between sequences.*

Observe that, in the literature, the definition of alignments is usually merged to the sequences of moves that encode a specific edit distance function.

Definition 9 (Sequence of Moves, Alignment Cost)**.** *Given a log trace $\sigma = \langle \sigma_1, \dots, \sigma_m \rangle \in L$, and a process model N,* alignments *of σ with N are given as* sequences of moves *$\langle (\sigma'_1, u'_1), \dots, (\sigma'_p, u'_p) \rangle$ with $p \leq m + n$ such that, for a given index i and a given run $u = \langle u_1, \dots, u_n \rangle \in Runs(N)$:*

- *each move* (σ'_i, u'_i) *is either: a synchronous move* (σ_j, u_k) *with* $\sigma_j = u_k$*, a log move* $(\sigma_j, \gg)$*, which represents the deletion of* σ_j *in* σ*, or a model move* $(\gg, u_k)$*, which represents the insertion of* u_k *in* σ*, where* $j \in \{1, \ldots, m\}$ *and* $k \in \{1, \ldots, n\}$*;*
- *the left projection* $\langle \sigma'_1, \ldots, \sigma'_p \rangle$ *of the alignment to* $\mathcal{A}^*$ *(which drops the occurrences of* $\gg$*), yields* σ*;*
- *the right projection* $\langle u'_1, \ldots, u'_p \rangle$ *of the alignment to* $\mathcal{A}^*$*(which drops the occurrences of* $\gg$*), yields* u*.*

The alignment cost *or* distance *is the count of non-synchronous moves, expect for silent transitions. Optimal sequences of moves are the ones that minimize the alignment cost given* σ *and* N*.*

Example 2.2.1 (Alignment, Sequence of Moves)**.** *For the log trace* $\sigma = \langle open, wait, write, close \rangle$ *and the process model* N *of Figure 7.3,* $u = \langle open, write, \tau, read, close \rangle$ *is a run of* N *optimizing the alignment cost, i.e.,* u *is an alignment of* σ *to* N*. The table below shows the sequences of moves between* σ *and* u*.*

σ	*open*	*wait*	*write*	$\gg$	$\gg$	*close*
u	*open*	$\gg$	*write*	τ	*read*	*close*

Since the sequence of moves contains three non-synchronous moves but one is a silent label, the alignment cost is 2.

We separate this structured form of alignment in order to refer to the known distances and allow the conformance artefacts to be delineated with other distance functions.

2.2.2 Distances

Def. 9 of the sequence of moves and alignment cost is a structured form of the Levenshtein edit distance where the synchronization is not allowed. Moves correspond to the edits in the distance.

Definition 10 (Levenshtein Edit distance)**.** *The* Levenshtein Edit Distance $\mathcal{L}(u, v)$ *between two sequences* u *and* $v \in \Sigma^*$ *is the minimal number of edits needed to transform* u *to* v*.*

$$\mathcal{L}(u,v) \stackrel{\text{def}}{=} \begin{cases} \mathcal{L}(\langle\rangle, \langle\rangle) = 0 \\ \mathcal{L}(u, \langle\rangle) = |u| \\ \mathcal{L}(\langle\rangle, v) = |v| \\ \mathcal{L}(a.u', b.v') = \mathcal{L}(u', v') & \text{if } (a = b) \\ \mathcal{L}(a.u', b.v') = \min \begin{cases} \mathcal{L}(a.u', v) + 1, \\ \mathcal{L}(u, b.v') + 1 \end{cases} & \text{otherwise.} \end{cases} \tag{2.1}$$

Edits can be deletions or additions of an activity in sequence.

Insertions and deletions correspond to the log and model moves.

Example 2.2.2 (Edit Distance). *Considering $\sigma = \langle open, wait, write, close\rangle$ and $u = \langle open, write, read, \tau, close\rangle$ the number of edits to transform σ to u is 2, where τ has a free edit. The activity wait has to be removed and the activity read inserted. Then the two sequences are at distance 2.*

Other distances have been tackled in the literature but their definition is less appropriate to process alignments. We illustrate it with the Hamming distance given below.

Definition 11 (Hamming distance). *The* Hamming Distance $\mathcal{H}(u, v)$ *between two sequences u and $v \in \Sigma^*$ is the is the number of positions at which the content are different in u and v.*

$$\begin{cases} \mathcal{H}(\langle\rangle, \langle\rangle) = 0 \\ \mathcal{H}(u, \langle\rangle) = |u| \\ \mathcal{H}(\langle\rangle, v) = |v| \\ \mathcal{H}(a.u', b.v') = \begin{cases} \mathcal{H}(u', v'), & \text{if } (a = b) \\ \mathcal{H}(u', v') + 1 & \text{otherwise.} \end{cases} \end{cases} \tag{2.2}$$

Example 2.2.3 (Hamming Distance). *Considering sequences $\sigma' = \langle write, read, close\rangle$ and $u = \langle open, write, read, \tau, close\rangle$, the Hamming distance is 4 as activities are different in all positions but τ offers a free edit. The Levenshtein distance between σ' and u is 1 because it removes activity open which allow to align all the other activities. The Hamming distance disabled the correspondences between σ and u which is not much appropriate for comparing process executions.*

2.2.3 Synchronous Product

The main methods of the literature to compute optimal alignments are Dijkstra-based algorithms which often implies the construction of the *Synchronous Product* between the given process model and a sequential Petri net representing the log trace [4].

Definition 12 (Synchronous Product for Alignments). *For a process model $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$ and a log trace $\sigma = \langle \sigma_1, \ldots, \sigma_m\rangle \in \Sigma^*$, the Synchronous Product used for computing alignments is the Petri net $SN = \langle P_{SN}, T_{SN}, F_{SN}, m_{SN_0}, m_{SN_f}, (\Sigma \cup \{\gg\})^2, \lambda_{SN}\rangle$ defined as:*

- *$N_\sigma = \langle P_\sigma, T_\sigma, F_\sigma, m_{\sigma 0}, m_{\sigma f}, \Sigma, \lambda_\sigma\rangle$ is a translation of σ to a sequential Petri net with: $P_\sigma = \{P_{\sigma_0}, \ldots P_{\sigma_m}\}$, $T_\sigma = \{t_{\sigma_i} = \lambda_\sigma(\sigma_i) \mid i \in \{1, \ldots, m\}\}$, $F_\sigma = \{(P_{\sigma_{i-1}}, t_{\sigma_i}), (t_{\sigma_i}, P_{\sigma_i}) \mid i \in \{1, \ldots, m\}\}$, $m_{\sigma_0} = \{P_{\sigma 0} : 1\}$, $m_{\sigma f} = \{P_{\sigma_m} : 1\}$,*

- $P_{SN} = P \cup P_\sigma$
- $T_{SN} = T^{\gg} \cup T_\sigma^{\gg} \cup T_S$, *where* $T^{\gg} = \{(\gg, t) \mid t \in T\}$ *represents the model moves,* $T_\sigma^{\gg} = \{(t, \gg) \mid t \in T_\sigma\}$ *represents the log moves,* $T_S = \{(t_1, t_2) \mid t_1 \in T, t_2 \in T_\sigma$ *and* $\lambda(t_1) = \lambda_\sigma(t_2)\}$ *represents the synchronous moves,*
- $F_{SN} = F \cup F\sigma \cup \{(P_i, t_i) \mid t_i = (t_1, t_2) \in T_{SN},\ t_1 \neq \gg,\ t_2 \neq \gg,\ P_i \in {}^\bullet t_1 \cap {}^\bullet t_2\}$ $\cup \{(t_i, P_i) \mid t_i = (t_1, t_2) \in T_{SN},\ t_1 \neq \gg,\ t_2 \neq \gg,\ P_i \in t_1{}^\bullet \cap t_2{}^\bullet\}$
- $m_{SN_0} = m_0 \cup m_{\sigma_0}$,
- $m_{SN_f} = m_f \cup m_{\sigma_f}$,
- λ_{SN} *maps every* $t \in T_{SN}$ *to its move.*

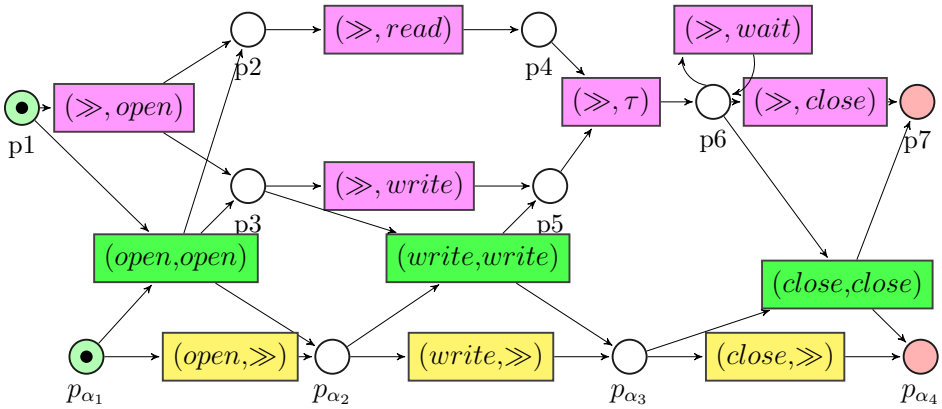


Figure 2.3: A Synchronous Product SN of Process Model N Given in Fig. 7.3 and the Log Trace $\sigma = \langle open, write, close \rangle$. Transitions in purple correspond to the model moves, green transitions are the synchronous moves and yellow transitions the log moves.

Example 2.2.4 (Synchronous product)**.** *Fig 2.3 shows an example of synchronous product. Transitions labeled with* $(open, open)$, $(write, write)$ *and* $(close, close)$ *correspond to synchronous moves and do not cost. Other transitions cost 1 (except* $(\gg, \tau)$*).*

The Dijkstra-based algorithm for finding optimal alignments, explores the reachability graph of the synchronous product. Weights are given by the transitions fired to reach the markings, according to the type of move that they represent. The best firing sequences found for reaching a marking is the less costly one.

As we are using easy-sound Petri nets as process models, the Synchronous Products for Alignments are easy-sound which implies termination of the Dijkstra algorithm with the condition to reach the final marking m_{SN_f} [142].

2.2.4 Alignment-based Fitness

Alignment is commonly used for measuring the fitness of a process model with regards to the log traces. Every log trace is aligned to the process model in order to obtain the alignment cost, or distance, between the trace and one of its optimal alignment of the model.

We compute the fitness of a process model with regards to a trace as follows:

$$\text{fitness}(\sigma, N) = 1 - \frac{\min_{u \in N} \text{dist}(\sigma, u)}{|\sigma| + \min\limits_{u' \in Runs(N)} |u'|} \tag{2.3}$$

where $\min_{u \in M} \text{dist}(\sigma, u)$ gives the optimal cost of aligning σ with M using a run u. Then, the fitness of the model is the average of the fitness of the traces for this model.

A trace is said to be *fitting* when its fitness is 1, i.e. when its optimal alignment has a cost of 0. We define the fitness of a process model M with regards to a log L to be the average of the fitness of M with regards to each log trace of L.

Example 2.2.5 (Fitness)**.** *The trace fitness of model N shown in Fig. 7.3 and $\sigma = \langle open, wait, write, close \rangle$ is given by:*

$$\mathit{fitness}(\sigma, N) = 1 - \frac{2}{4+5} = 0.77$$

2.3 Formal Methods

The present thesis claims to belong to the subfield of conformance checking which implies interest in complexity and optimality. In this section we recall complexity of a knonw Petri net problem and the SAT definition of Petri nets.

2.3.1 Petri Nets and Complexity

The ability of Petri nets to offer modeling for concurrency, choice and repetitions contained in a language enrolls an interest in decidability and complexity of the verification problems involved in these nets. A impressive collection of questions have been explored from late seventies to early eighties, engaging different classes of Petri nets [69]. Today most of them are resolved [54, 53]. We present below the reachability problem, a baseline of complexity problems in Petri nets, that we also use to identify the complexity of our methods.

Definition 13 (Reachability in Petri nets)**.** *The* reachability problem *for Petri nets consists of deciding, given a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ and a marking ω of N, if ω can be reached from m_0, i.e., if there exist a firing sequence $\langle t_1 \dots t_n \rangle$ such that $m_0[t_1 \dots t_n \rangle \omega$.*

The problem is known to be decidable, but non-elementary [37], and still PSPACE-complete for safe Petri nets. But the complexity trivially drops to NP-complete if a bound l is given (with l an integer coded in unary) on the length of u. NP-hardness can be obtained by reduction from the problem of reachability in a safe acyclic Petri net, known to be NP-complete [34].

The decidability of alignments is reducted to the problem of reachability of Petri nets by using the synchronous product and its final marking. Alignment computation is then also NP-complete when a bounded size of runs is given as input in unary [31, 19].

2.3.2 SAT Encoding

The Boolean satisfiability (or SAT) problem, is the problem of determining, for a given Boolean formula, if there exists a combination of assignments to the variables that satisfies it. For instance, in the case of marking reachability, a SAT formula would encode the following question: *Is the marking m of N reachable ?* In other words, for this example, we are looking for a Conjunctive Normal Formula (CNF) that encodes the token game of the process model N. As SAT formulas encode problems as they are, solving them gives optimality.

Petri Nets as SAT Instances

The SAT encoding of Petri net is not a novelty of the literature. We recall the formulas given in [31].

For a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ and n the size of the full runs, the boolean variables $m_{i,p}$, with $i \in \{0..n\}$ and $p \in P$, represent the marking at instant i. Then, the variables $\mu_{i,t}$ encode a firing transition $t \in T$ at instant $i \in \{0..n\}$: The following constraints encode the semantics of the Petri net.

- Initial marking:
$$(\bigwedge_{p \in m_0} m_{0,p}) \wedge (\bigwedge_{p \in P \setminus m_0} \neg m_{0,p}) \tag{2.4}$$
- Final marking:
$$(\bigwedge_{p \in m_f} M_{n,p}) \wedge (\bigwedge_{p \in P \setminus m_f} \neg m_{n,p}) \tag{2.5}$$
- One and only one t_i for each i:
$$\bigwedge_{i=1}^{n} \bigvee_{t \in T} (\mu_{i,t} \wedge \bigwedge_{t' \in T \setminus \{t\}} \neg \mu_{i,t'}) \tag{2.6}$$
- The transitions are enabled when they fire:
$$\bigwedge_{i=1}^{n} \bigwedge_{t \in T} (\mu_{i,t} \implies \bigwedge_{p \in {}^\bullet t} m_{i-1,p}) \tag{2.7}$$

- Token game (for safe Petri nets):

$$\bigwedge_{i=1}^{n} \bigwedge_{t \in T} \bigwedge_{p \in t^\bullet} (\mu_{i,t} \implies m_{i,p}) \tag{2.8}$$
$$\bigwedge_{i=1}^{n} \bigwedge_{t \in T} \bigwedge_{p \in {}^\bullet t \setminus t^\bullet} (\mu_{i,t} \implies \neg m_{i,p}) \tag{2.9}$$
$$\bigwedge_{i=1}^{n} \bigwedge_{t \in T} \bigwedge_{p \in P, p \notin {}^\bullet t, p \notin t^\bullet} (\mu_{i,t} \implies (m_{i,p} \iff m_{i-1,p})) \tag{2.10}$$

Example 2.3.1 (SAT encoding)**.** *Consider the model of Fig. 7.3. At the initialization, place* $p1$ *has a token. This information is encoded by* $m_{0,p1} = \text{true}$*. All the other places do not have a token at this instant, this is described by axiom (2.4). At the first instant, only transition* $t1$ *labeled by open can fire. We then have* $\mu_{1,t1} = \text{true}$*. All the others transitions do not fire at this instant. For instance* $\mu_{1,t2}$ *is false, i.e., transition* $t2$ *for activity read does not fire at instant 1. This behavior is defined by axiom (2.6). The last axioms define the token game.*

The presented clauses are then grouped in a conjunction and the entire formula is transformed to a *Conjunctive Normal Form* (CNF) to be handled by the solvers of the literature. Henceforth, process model' semantics can be solved by a SAT formula.

MinSAT/MaxSAT Instances and Weigthed Clauses

MinSAT problem is the problem of finding boolean assignments that minimizes the number of truth clauses in a CNF formula. In opposite, MaxSAT problems look for a solution that maximizes the number of truth clauses. They are generalizations of SAT problems, which ask whether there exists a truth assignment that makes clauses true.

Example 2.3.2 (MaxSAT)**.** *Consider the following CNF formula:*

$$(x_0 \wedge x_1) \vee (\neg x_0 \wedge x_1) \vee (x_0 \wedge \neg x_1) \vee (\neg x_0 \wedge \neg x_1)$$

The formula is not satisfiable. However, we can get the minimal number of truth clauses (one) and the maximal number of truth clauses (three).

Weigthed MinSAT and weighted MaxSAT problems generalize the idea which non-negative weights assigned to the clauses of the formula. Two types of clauses can then be distinguished: hard clauses and soft clauses. Hard clauses have to be satisfied whether soft clauses are tackled by the minimization or maximization of the truth assignments.

Technical Details

Nicely, the recent library **Pysat** in Python provides a great access to MinSAT/MaxSAT solvers.

The uses of SAT encodings aims at finding optimal solutions. Recent studies focus on SAT implementation for Data Mining algorithms, in order to satisfy all the constrains and

get optima [82, 38]. In this thesis, all the methods are first presented and implemented as SAT, MinSAT or MaxSAT problems in order to push a new family of algorithmic methods for conformance checking in the line of [31, 19]. However, these aforementioned works mostly consider Hamming distance between log traces and process models, which is usually considered less appropriate than edit distance (see example 2.2.3).

Chapter 3

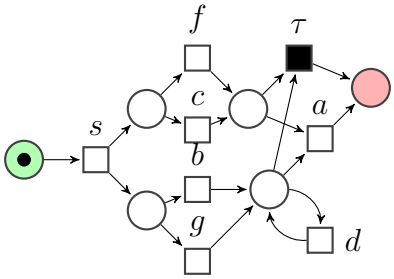
Multi-Alignments: Conformance Checking Artefacts for Model-based Representations of Logs and Sub-Logs

Chapter Overview

The first contribution of this thesis is the computation of multi-alignments, i.e., a conformance checking artefact that represents a set of log sequences. The artefact is introduced in Section 3.1. Section 3.2 gives the related work. Sections 3.3 and 3.4 provide respectively a MinSAT-based algorithm and an A*-based algorithm for computing the artefact. Firstly elaborated for alignment computation, those algorithms are flexible to the different conformance checking artefacts. The first algorithm allows to compute optimal multi-alignment for the first time. However, due to the complexity of the obtained MinSAT formula, this algorithm is restricted to artificial or small instances. We elaborated an A*-based algorithm to approximate multi-alignments. This later algorithm uses a novel distance function entitled the *discounted edit distance*. In Section 3.5, we show several experiments of the two algorithms along with comparisons to the state-of-the-art methods for alignments. We also present a case study where multi-alignments suit well for model-based variants. We conclude the chapter by giving the limits and bridge to the clustering methods for finding good representatives.

3.1 Introduction

Multi-alignments were introduced in [32] as a generalization of alignments. Instead of aligning a trace to a run of a process model, multi-alignments allow to align a set of log traces to a joint run of the model. The aim of this concept is to get an overview of several traces with respect to a process model, i.e., a *model-based trace variant*.



(a) Process Model $N1$

$\{\langle s, f, b, a\rangle,$
$\langle s, b, f\rangle,$
$\langle s, c, b, a\rangle,$
$\langle s, g, c, a, d\rangle,$
$\langle s, d, d\rangle\}$

(b) Log $L1$

Figure 3.1: Drawing Example for Multi-alignments.

Definition 14 (Multi-alignment)**.** *Given a finite collection L of log traces and a model N, an (optimal)* multi-alignment *of L to N is a full run $u \in Runs(N)$ which minimizes the quantity* $\max_{\sigma \in L} dist(\sigma, u)$ *that represents the maximal distance to any log trace, i.e.,*

$$\min_{u \in Runs(N)} \max_{\sigma \in L} dist(\sigma, u) \tag{3.1}$$

where dist is a distance between sequences.

The preferred distance *dist* for multi-alignment is, like for alignments, the Levenshtein edit distance. The collection L aforementioned in the definition can be the entire log. However, in practice, the use of multi-alignment fits well when the selected log sequences are already related and the process owner looks for their model-based representation.

Example 3.1.1 (Multi-alignments)**.** *For the process model $N1$ and the log $L1$ a multi-alignment is $\langle s, c, b, \tau\rangle$ of maximal distance 4 to a log trace, i.e., there is no run of N_1 such that the maximal distance to any log trace is lower than 4.*

Multi-Alignment	*Trace*	*Distance*
$\langle s, c, b, \tau\rangle$	$\langle s, f, b, a\rangle$	*3*
	$\langle s, b, f\rangle$	*2*
	$\langle s, c, b, a\rangle$	*1*
	$\langle s, g, c, a, d\rangle$	*4*
	$\langle s, d, d\rangle$	*4*

As for comparison, we show classical alignments of the log traces.

Alignment	*Trace*	*Distance*
$\langle s, f, b, a\rangle$	$\langle s, f, b, a\rangle$	*0*
$\langle s, b, f, \tau\rangle$	$\langle s, b, f\rangle$	*0*
$\langle s, c, b, a\rangle$	$\langle s, c, b, a\rangle$	*0*
$\langle s, g, c, d, d, \tau\rangle$	$\langle s, g, c, a, d\rangle$	*1*
$\langle s, b, c, d, d, \tau\rangle$	$\langle s, d, d\rangle$	*2*

Despite being less accurate to the classical alignments, which provide a result per trace, multi-alignments give a modeled "median" view of the log traces. When the maximal distance is low, multi-alignments suit well for model-based trace variant.

Example 3.1.2 (Multi-alignments as Model-based Variants)**.** *For instance, for* $L1' = \{\langle s, f, b, a\rangle, \langle s, c, b, a\rangle, \langle s, c, b\rangle\}$*, we find the multi-alignment* $\langle s, c, b, a\rangle$ *of maximal distance 1 which represents well all the log traces.*

As it happens for alignments, there is not only one optimal multi-alignment for a given log and model.

Example 3.1.3 (Several Optimal Multi-alignments)**.** *The modeled sequence* $\langle s, b, c, d, a\rangle$ *is also at maximal distance 4 to any log trace of* $L1$*.*

Given the observation developed in Section 2.2 about the definition of alignments in the literature, multi-alignments can also be enriched with the sequences of moves. The discovered run that holds for multi-alignment is aligned to each log trace separately. Thus, for each trace, one can build the sequences of moves with the found multi-alignment. The number of non-free moves per sequence is then bounded by the distance of the multi-alignment to the log.

Example 3.1.4 (Sequences of Moves in the case of Mutli-alignments)**.** *For* $u = \langle s, c, b, \tau\rangle$ *a multi-alignment of* $N1$ *for* $L1$*, we give an example of sequences of moves for this multi-alignment* u *and every log trace of* $L1$*:*

$\langle s, f, b, a\rangle$	s	f	$\gg$	b	$\gg$	a
u	s	$\gg$	c	b	τ	$\gg$

$\langle s, b, f\rangle$	s	$\gg$	b	$\gg$	f
u	s	c	b	τ	$\gg$

$\langle s, c, b, a\rangle$	s	c	b	$\gg$	a
u	s	c	b	τ	$\gg$

$\langle s, g, c, a, d\rangle$	s	g	c	$\gg$	a	d	$\gg$
u	s	$\gg$	c	b	$\gg$	$\gg$	τ

$\langle s, d, d\rangle$	s	$\gg$	$\gg$	$\gg$	d	d
u	s	c	b	τ	$\gg$	$\gg$

These sequences of moves for multi-alignment is not in the scope of interest of this thesis, and we favor the distance notation between process model runs and log traces.

An algorithm for computing multi-alignments with Hamming distance is proposed in [32]. However, this distance is less appropriate than edit distance in Process Mining (see Example 2.2.3). In this chapter, we fill the gap by providing two algorithms for computing multi-alignments by using some edit distances:

- **a MinSAT-based algorithm** which encodes the definition as it is to get optima

- **an A*-based algorithm** which contains several heuristics to get fast multi-alignment approximations for real-life instances

Each multi-alignment algorithm is preceded by the alignment-based version that we can compare with the state-of-the-art methods.

3.2 Related Work

In this related work section, the differences and similarities of sequence alignment in bioinformatics, where no process model is involved, are presented. Dealing with a process model is a big challenge of Process Mining already when it comes to aligning a single trace only. The second part of this section then focuses on alignment computation and approximation which is core interest of this chapter. Indeed, multi-alignments are a generalization of alignments and our algorithms can be applied for aligning a single log trace to a process model. Moreover, there is no other methods in the literature for computing multi-alignment apart from the ones presented in this thesis.

3.2.1 Multiple Sequence Alignment in Bioinformatics

The notion of multi-alignments in Process Mining appears in 2017 in [32] and is not to confuse with *Multiple Sequence Alignment* (MSA), one of the most important challenges in bioinformatics [26, 29]. Despite its common interest in relating a set of sequences through a unique alignment, MSA does not involve a process model whom language is not necessarily bounded. Moreover, the maximization problem of MSA focuses on the sum of common pairs between all the sequences while multi-alignments aims at finding a unique modeled sequence close to all the log traces separately, i.e., there is no sequence-to-sequence relation. However, the two fields stay close and we can observe works in MSA interested in "median sequence" [64], and partial order and markov chain representations [72, 51].

About implementation, Prestwich et al. present in [95] a work close to our where a graph, representing the symbol alignment between sequences, is encoded in SAT. This graph reminds the synchronous product between Petri nets because its aim is also to align symbols with weighted edges where, in the synchronous product, we use weighted transitions.

3.2.2 Computation of Alignments

The seminal work in [4] proposed the notion of alignment in Process Mining and developed a technique based on A* to compute optimal alignments for a particular class of process models. Improvements of this approach have been presented recently in different papers [126, 128]. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory

footprint) to provide all optimal alignments. Alternatives to A* have appeared in recent years: in the approach presented in [43], the alignment problem is mapped as an *automated planning* instance. Automata-based techniques have also appeared [101, 74]. The techniques in [101] (recently extended in [102]) rely on state-space exploration and determination of the automata corresponding to both the event log and the process model, whilst the technique in [74] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [111] presented the notion of *approximate* alignment to alleviate the computational demands by proposing a recursive paradigm on the basis of the structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. A similar approach which can always guarantee a solution and heavily uses the resolution of *Integer Linear Programming* (ILP) and marking equation in combination with a bounded backtracking is presented in [126].

Decomposition techniques have been presented in [122, 85, 136] that, instead of computing optimal alignments, they focus on the *crucial problem* of whether a given trace fits or not a process model. These techniques vertically decompose the process model into pieces satisfying certain conditions (so only *valid* decomposition [122], which satisfy restrictive conditions on the labels and connections forming a decomposition, guarantee the derivation of a real alignment). Later on, the notion of *recomposition* has been proposed on top of decomposition techniques, in order to obtain optimal alignments whenever possible by iterating the decomposition methods when the required conditions do not hold [73]. Decomposition techniques brings us to the recent work of [131] which presents an online alignment technique with a window-based backwards exploration.

Alternatively, the technique in [112] presents a framework to reduce a process model and the event log accordingly, with the goal of alleviating the computation of alignments. The obtained alignment, called *macro-alignment* since some of the positions are high-level elements, is expanded based on the information gathered during the initial reduction. Techniques using local search have recently been also proposed very recently [113].

In [20], the authors propose to disable the computation of alignments after a learning stage. The idea is to use machine learning techniques to predict if a given trace is fitting to a process model.

Finally, work of [15] proposes a slightly different goal for alignment. The aim of their algorithm is to maximize the number of synchronous moves in the alignment where alignment usually minimizes the number of asynchronous moves.

3.3 MinSAT Encoding for Computing Multi-alignments

The optimal method to compute multi-alignment with the Levenshtein edit distance is the MinSAT encoding of the problem which is first set as a SAT problem as follows: *Does a multi-alignment exist between the log traces L and the model N for a maximal distance d?*

To encode this problem, we first introduce our SAT encoding of the edit distance between two sequences. This building block serves for alignments in general. We provide

the details to relate process models and traces for alignments and extend it to multi-alignments.

3.3.1 SAT Encoding of Edit Distance

Our encoding of the Levenshtein edit distance is based on the same relations that are used by the classical dynamic programming recursive algorithm for computing the distance between two sequences $u = \langle u_1, \ldots, u_n \rangle$ and $v = \langle v_1, \ldots, v_m \rangle$:

$$\begin{cases} dist(\langle u_1, \ldots, u_i \rangle, \epsilon) = i \\ dist(\epsilon, \langle v_1, \ldots, v_j \rangle) = j \\ dist(\langle u_1, \ldots, u_{i+1} \rangle, \langle v_1, \ldots, v_{j+1} \rangle) = \\ \quad \begin{cases} dist(\langle u_1, \ldots, u_i \rangle, \langle v_1, \ldots, v_j \rangle) & \text{if } u_{i+1} = v_{j+1} \\ 1 + \min(dist(\langle u_1, \ldots, u_{i+1} \rangle, \langle v_1, \ldots, v_j \rangle), & \\ \qquad dist(\langle u_1, \ldots, u_i \rangle, \langle v_1, \ldots, v_{j+1} \rangle)) & \text{if } u_{i+1} \neq v_{j+1} \end{cases} \end{cases} \tag{3.2}$$

We encode this computation in a SAT formula ϕ over variables $\delta_{i,j,d}$, for $i = 0, \ldots, n$, $j = 0, \ldots, m$ and $d = 0, \ldots, n + m$. The formula ϕ has exactly one solution, in which each variable $\delta_{i,j,d}$ is `true` iff $dist(\langle u_1 \ldots u_i \rangle, \langle v_1 \ldots v_j \rangle) \geq d$.

In order to test equality between the u_{i+1} and v_{j+1}, we use variables $\lambda_{i+1,a}$ and $\lambda'_{j+1,a}$, for $i = 0, \ldots, n$, $j = 0, \ldots, m$ and $a \in \Sigma$, which holds for having activity a at instant i translated by $\lambda_{i+1,a}$ is `true` iff $u_{i+1} = a$, and $\lambda'_{j+1,a}$ is `true` iff $v_{j+1} = a$. Hence, the test $u_{i+1} = v_{j+1}$ becomes:

$$\bigvee_{a \in \Sigma} (\lambda_{i+1,a} \wedge \lambda'_{j+1,a}) \tag{3.3}$$

For readability of the formulas, we refer to this coding by $[u_{i+1} = v_{j+1}]$. Similarly, we write $[u_{i+1} \neq v_{j+1}]$.

In the following, we describe the different clauses of the formula ϕ of our SAT encoding of the edit distance.

$$\delta_{0,0,0} \quad \wedge \quad \bigwedge_{d>0} \neg \delta_{0,0,d} \tag{3.4}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Leftrightarrow \delta_{i,0,d}) \tag{3.5}$$

$$\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Leftrightarrow \delta_{0,j,d}) \tag{3.6}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Leftrightarrow \delta_{i,j,d}) \tag{3.7}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Leftrightarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{3.8}$$

The conjunction of all the constraints get us a SAT formula of the Leventshein distance.

Example 3.3.1 (SAT encoding of the Levenshtein edit distance)**.** *At instants $i = 1$ and $j = 1$ of sequences $u = \langle s, g, c\rangle$ and $v = \langle s, b, c, a\rangle$, the activities are the same (s), then, by Eq. (3.4), the distance is only higher or equal to 0: $(u_1 = v_1) \Rightarrow (\delta_{1,1,0} \Leftrightarrow \delta_{0,0,0})$.*
However at instants $i = 2$ and $j = 2$, the letters u_2 and v_2 are different. A step before, $\delta_{1,2,1}$ and $\delta_{2,1,1}$ are true because of the length of the sub-sequences. Then, by (5), the distance at instants $i = 2$ and $j = 2$ is higher or equal to 2: $\delta_{2,2,2}$. The result is correct because the edit distance costs the deletion of g and the addition of b to transform u to v.

3.3.2 SAT Encoding of Alignments

The above clauses are considered in the SAT implementation of alignments. A log trace σ is a sequence of activities and is encoded as presented in the previous section with boolean variables $\lambda_{i,a}$ where $i \in \{1, \ldots, n\}$ and $a \in \Sigma$. SAT encoding of process models has been recalled in Background's section. The fired transition at instant i and label a is given by $\mu_{i,a}$.

The last series of constraints that is needed to be appended is the relation of the fired transitions to the activities of the log trace:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \bigvee_{a \in \Sigma} (\lambda_{i,a} \iff \mu_{j,a}) \tag{3.9}$$

Example 3.3.2 (SAT variables explanation for alignment)**.** *All the full runs of the process model of Fig. 3.1a contain a "s" at the first instant. So the variable $\mu_{1,s}$ is true. If the log trace is $\sigma = \langle s, f, b, a\rangle$ then, $\lambda_{1,s}$ is true which implies $\delta_{1,1,0}$ by (3.4).*

Technical Details

In order to consider different sizes of traces and different sizes of runs, we added a loop on a *wait* activity on the final marking of the model. The SAT encoding of the edit distance is adjusted so that skipping a **wait** activity does not increment the distance between sequences.

Minimization of the Distance for Optimal Alignment

The conjunction of the previous clauses for the full runs of the model and the edit distance to a given log trace σ, gives a formula which has one solution per full run of the model. With each solution, the values of the $\delta_{n,m,d}$ determine the edit distance between the corresponding

modeled sequence and σ. Our goal for optimal alignment is to minimize this distance, which corresponds to the number of variables assigned to `true` among the $\delta_{n,m,d}$. This problem is then a MinSAT instance where the minimization objective is the sum $\sum_d 1 \times \delta_{n,m,d}$.

We associated clauses of $\delta_{n,m,d}$ boolean variables to a weight of 1 while all the other clauses are hard clauses. Then, while resolving the SAT formula, the minimization of the weights of the variables $\delta_{n,m,d}$ forces to assign those variables to `false`. Only the required $\delta_{n,m,d}$ will be kept and the minimization of the distance can be found, i.e the cost of an optimal alignment.

3.3.3 Application for Multi-Alignments

The SAT implementation of multi-alignment requires us to duplicate the variables $\lambda_{i,a}$, now noted $\lambda^{\sigma}_{i,a}$, that represent actions in the log traces $\sigma \in L$ and the variables $\delta^{\sigma}_{i,j,d}$ that measure the edit distance to the run for each trace $\sigma \in L$.

Multi-alignment considers the run of the model that minimizes its maximal distance to the log traces. To produce the general distance of the run and all the traces, we introduce novel variables Δ_d and the following axiom:

$$\bigvee_d \left(\bigwedge_\sigma \delta^{\sigma}_{n,|\sigma|,d} \Leftrightarrow \Delta_d \right) \tag{3.10}$$

where n is the size of the run. The Δ_d variables define the distances d for which all the traces verify this distance to the run of the model. The minimization objective for multi-alignment is then : $\sum_d 1 \times \Delta_d$.

Example 3.3.3 (Delta Boolean Variables)**.** *We computed the multi-alignment of the model and the full log of Fig. 3.1a. The optimal multi-alignment is the full run $\langle s, c, b, \tau \rangle$ which is at distance $d \leq 4$ to all the log traces. Then the maximal d such that a $\delta^{\sigma}_{n,m,d}$ is true is $d = 4$.*

Technical Details

Observe that Section 5.3.1, which tackles anti-alignments, benefits of the same algorithm thanks to a modification on weights.

Minimization of Maximal Distance versus Sum of Distances

In [18], it was presented a variant of multi-alignment that aims at optimizing the *sum* of the distances between the run and the traces instead of the maximal distance between the run and the traces.

Although in a way, optimizing the sum of distances tends to obtain in general reasonable witness of multi-alignments, in some situations they can fail in representing a good solution.

The SAT formula does not require variables Δ_d introduced in the previous minimization. Similarly to alignments, the optimal multi-alignment is found by minimizing objective: $\sum_d \sum_{\sigma \in L} 1 \times \delta^{\sigma}_{n,|\sigma|,d}$.

The two minimizations presented in this section give different definitions, that may not be used in the same context. To convince readers, we propose the following comparison.

Example 3.3.4 (Comparison of multi-alignments minimization types)**.** *The following table shows the two multi-alignment types for model $N1$ and log $L1$ of Fig. 3.1.*

Method	*Multi-Alignment*	*Maximal Distance*	*Sum of Distances*
Maximizing Minimal Distance	$\langle s, c, b, \tau \rangle$	*4*	*14*
Minimizing Sum of Distances	$\langle s, c, b, a \rangle$	*5*	*13*

We observe that the minimization of the sum returns the multi-alignment $\langle s, c, b, a \rangle$ which is a behavior that appears in the log. Then, for this trace, the distance is null which helps for minimizing the sum of distances. However, the multi-alignment $\langle s, c, b, \tau \rangle$ gives a better "median" view of the log sequences.

3.3.4 Formula Reduction

For multi-alignment, the SAT edit distance formula is duplicated by the number of log traces, which generates thousands of variables (see graphs of Fig. 3.6). In this section we present a reduction of the formula. The main idea is to keep only one direction of the double implications of the SAT encoding of the Levenshtein edit distance. As double implications create clauses in the SAT formula, we improve the size of the formula and its resolution.

As seen in Section 3.3.1, the edit distance between two sequences corresponds to the number of variables assigned to `true` among the $\delta_{n,m,d}$ in the (unique) solution s of the formula ϕ. Let us denote this value $val(s)$. Now, when searching for an alignment to a log trace (consider only alignment to one trace for simplicity), we combine the formula ϕ with a formula ψ which encodes a set of runs of the model, and look for the solution s of the combined formula $\Phi \equiv \phi \wedge \psi$ which minimizes $val(s)$. This minimization amounts to filter the set of solutions of Φ. Here, we show that, relying on this filtering by the minimization, we can reduce the formula Φ (that we now denote $\Phi_{\Leftrightarrow}$) constructed from the formula ϕ of Section 3.3.1 (now denoted $\phi_{\Leftrightarrow}$) into a simpler formula $\Phi_{\Leftarrow}$ constructed from a reduced version of $\phi_{\Leftrightarrow}$, denoted $\phi_{\Leftarrow}$ and defined as:

$$\delta_{0,0,0} \quad \wedge \quad \bigwedge_{d>0} \neg\delta_{0,0,d} \tag{3.11}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Leftarrow \delta_{i,0,d}) \tag{3.12}$$

$$\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Leftarrow \delta_{0,j,d}) \tag{3.13}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Leftarrow \delta_{i,j,d}) \tag{3.14}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Leftarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{3.15}$$

Lemma 1. *The minimal value obtained by minimizing $val(s)$ over the solutions of $\Phi_{\Leftarrow}$ is equal to the minimal multi-alignment distance obtained using $\Phi_{\Leftrightarrow}$. Formally, denote $sol(\Phi_{\Leftrightarrow})$ (respectively $sol(\Phi_{\Leftarrow})$) the set of solutions of $\Phi_{\Leftrightarrow}$ (respectively $\Phi_{\Leftarrow}$):*

$$\min_{s \in sol(\Phi_{\Leftarrow})} val(s) = \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s') \; . \tag{3.16}$$

Proof. We represent every solution of a SAT formula as an application $s : Vars \rightarrow \{\texttt{true}, \texttt{false}\}$ where $Vars$ is the set of variables of the formula, so that $s(v)$ denotes the value assigned to variable v in s.

1. $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \geq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: As $\Phi_{\Leftarrow}$ is defined like $\Phi_{\Leftrightarrow}$ with less constrains, we have $\Phi_{\Leftrightarrow} \Rightarrow \Phi_{\Leftarrow}$, then $sol(\Phi_{\Leftrightarrow}) \subseteq sol(\Phi_{\Leftarrow})$ which implies $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \geq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$.

2. $\min_{s \in sol(\Phi_{\Leftarrow})} val(s) \leq \min_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: Let $s \in sol(\Phi_{\Leftarrow})$. We will show how to create $s' \in sol(\Phi_{\Leftrightarrow})$ such that $val(s') \geq val(s)$. We define s' as follows :

   - $\forall_{i \in \{0...n\},\ p \in P} \quad s'(m_{i,p}) := s(m_{i,p})$
   - $\forall_{i \in \{1...n\},\ a \in \Sigma} \quad s'(\mu_{i,a}) := s(\mu_{i,a})$
   - $\forall_{\sigma \in L,\ i \in \{1...|\sigma|\},\ a \in \Sigma} \quad s'(\lambda^{\sigma}_{i,a}) := s(\lambda^{\sigma}_{i,a})$

   and

   - $\forall_{\sigma \in L,\ i \in \{1...n\}, j \in \{1...|\sigma|\},\ d \in \{0...(n+|\sigma|)\}}$ $s'(\delta^{\sigma}_{i,j,d}) := (dist(\langle u_1, .., u_i \rangle, \langle \sigma_1, .., \sigma_j \rangle) \geq d)$ where *dist* is the edit distance. i.e., s' assigns the values for the variables $\delta^{\sigma}_{i,j,d}$ according to the exact edit distance while solution s represents under-approximation of the distances.

   We then demonstrate that s' is indeed a solution of $\Phi_{\Leftrightarrow}$:

   - variables $m_{i,p}$, $\mu_{i,a}$ and $\lambda^{\sigma}_{i,a}$ are assigned like in s which is a solution of $\Phi_{\Leftarrow}$ and those variables are not affected by the reduction.
   - variables $\delta^{\sigma}_{i,j,d}$ are defined using the edit distance which verifies axioms (3.4) to (3.8) of $\Phi_{\Leftrightarrow}$.

Finally, we show that $val(s) \leq val(s')$: As, for multi-alignment, we minimize the distances, let's demonstrate that $s'(\delta^{\sigma}_{i,j,d}) \Rightarrow s(\delta^{\sigma}_{i,j,d})$ for $\sigma \in L$, $i \in \{0,..,n\}$, $j \in \{0,..,|\sigma|\}$ and $d \in \{0,..,(n+|\sigma|)\}$.

Let $\sigma \in L$. We prove by induction on n that:

$$\forall_{i,j,i+j\leq n} \quad \forall_d \quad (dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i,j,d}).$$

- Initialization : for $n = 0$, which implies $i = 0$ and $j = 0$, we have:
  - $d = 0 \quad dist(\epsilon,\epsilon) = 0$ and $\delta^{\sigma}_{0,0,d} = \texttt{true}$.
    Then it verifies $(dist(\epsilon,\epsilon) \geq 0) \Rightarrow s(\delta^{\sigma}_{0,0,0})$
  - $\forall_{d>0}, \quad dist(\epsilon,\epsilon) < d$ and $\delta^{\sigma}_{0,0,d} = \texttt{false}$.
    Then it verifies $(dist(\epsilon,\epsilon) \geq d) \Rightarrow s(\delta^{\sigma}_{0,0,d})$.
- Induction step: Assuming that $(dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i,j,d})$ holds for all i,j,d such that $i+j \leq n$, we show that the implication still holds when $i+j = n+1$:
  - for $i = 0$ (i.e. $u = \epsilon$): assume $(dist(\epsilon, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d)$ is true. By definition of the edit distance, $dist(\epsilon, \langle \sigma_1,\ldots,\sigma_j\rangle) = 1 + dist(\epsilon, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle$, which implies $dist(\epsilon, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle) \geq d-1$. By the induction we know that $(dist(\epsilon, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle) \geq d-1) \Rightarrow s(\delta^{\sigma}_{0,j-1,d-1})$. And finally, since s satisfies axiom (3.12), we have $s(\delta^{\sigma}_{0,j,d})$.
  - for $j = 0$ (i.e. $\sigma = \epsilon$): the proof is similar to the previous case (with axiom (3.13).
  - for $u_i = \sigma_j$: assume $dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d$. By the induction hypothesis, $(dist(\langle u_1,\ldots,u_{i-1}\rangle, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle) \geq d) \Rightarrow s(\delta^{\sigma}_{i-1,j-1,d})$. Here, $dist(\langle u_1,\ldots,u_{i-1}\rangle, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle) = dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle)$ by definition of the edit distance. Then $s(\delta^{\sigma}_{i-1,j-1,d})$ holds, and, since s satisfies axiom (3.15), we have $s(\delta^{\sigma}_{i,j,d})$.
  - for $u_i \neq \sigma_j$: if $(dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d)$ is true, then, by definition of the edit distance, $(dist(\langle u_1,\ldots,u_{i-1}\rangle, \langle \sigma_1,\ldots,\sigma_j\rangle) \geq d-1)$ and $(dist(\langle u_1,\ldots,u_i\rangle, \langle \sigma_1,\ldots,\sigma_{j-1}\rangle) \geq d-1)$. As $i-1+j = n$ and $i+j-1 = n$, we use the induction hypothesis to get $s(\delta^{\sigma}_{i-1,j,d}) = \texttt{true}$ and $s(\delta^{\sigma}_{i,j-1,d}) = \texttt{true}$. From axiom (3.15), we obtain $s(\delta^{\sigma}_{i1,j,d}) = \texttt{true}$.

□

Technical Details

In Section.3.5.2, we show the reduction of the formula in term of CNF clauses for different sizes of the run and log sizes.

3.3.5 Heuristics for the SAT Encoding

Due to the complexity of the problem, we present two heuristics to approximate the conformance checking artefacts by using the MinSAT-based method.

Prefix Limitation

Alignment and multi-alignments are runs of the process model that minimize the maximal distance to the log traces. A first approximation is then to compute only a prefix of them.

Definition 15 (Prefixes of Petri net)**.** *A* Prefix of size n for a Petri net $N = \langle P, T, F, M_0, M_f, \Sigma, \lambda \rangle$ *is a firing sequence* $M_0[u_1\rangle \dots M_{n-1}[u_n\rangle M_n$*, starting from the initial marking* M_0 *and of maximal length* n*.*

Example 3.3.5 (Prefixes of Runs)**.** *The prefixes of size* 2 *for the model of Fig. 3.1a are* $\langle s, b\rangle, \langle s, f\rangle, \langle s, g\rangle$*, and* $\langle s, c\rangle$*.*

Similarly, the log traces are truncated.

Maximal Number of Edits

The SAT encoding of the Levenshtein distance between two words u and v, presented in Section 3.3.1, uses boolean variables $\delta_{i,j,d}$ for $i = 0, \dots, |u|$, $j = 0, \dots, |v|$ and $d = 0, \dots, |u| + |v|$. The maximal number of edits max_d heuristic is an approximation of the SAT formula that aims at reducing the size of the formula by removing some clauses. Optimal formula of Levenshtein distance between words u and v requires boolean variables for $|u| + |v|$ edits (parameter d of $\delta_{i,j,d}$). The maximal number of edits heuristic uses variables $\delta_{i,j,d}$ only with d smaller than some bounds $max\_d < |u| + |v|$. This way, it reduces also the size of the formula and the computation time to solve it.

The heuristic precisely encodes the computation of the following value $dist_{\max\_d}(u, v)$ defined by slightly modifying the definition of Levenshtein's distance (compare with the definition of $dist(u, v)$ in Section 3.3.1):

$$\begin{cases} dist_{max\_d}(\langle u_1, \dots, u_i\rangle, \epsilon) = \min(d, i) \\ dist_{max\_d}(\epsilon, \langle v_1, \dots, v_j\rangle) = \min(d, j) \\ dist_{max\_d}(\langle u_1, \dots, u_{i+1}\rangle, \langle v_1, \dots, v_{j+1}\rangle) = \\ \quad \begin{cases} dist_{max\_d}(\langle u_1, \dots, u_i\rangle, \langle v_1, \dots, v_j\rangle) & \text{if } u_{i+1} = v_{j+1} \\ \min(d, 1 + \min(dist_{max\_d}(\langle u_1, \dots, u_{i+1}\rangle, \langle v_1, \dots, v_j\rangle), & \\ \qquad dist_{max\_d}(\langle u_1, \dots, u_i\rangle, \langle v_1, \dots, v_{j+1}\rangle))) & \text{if } u_{i+1} \neq v_{j+1} \end{cases} \end{cases} \tag{3.17}$$

Lemma 2. *For every* u *and* v*,* $dist_{max\_d}(u, v) \leq dist(u, v)$*. Moreover, if* $dist(u, v) \leq d$ *then* $dist_{max\_d}(u, v) = dist(u, v)$*.*

Proof. Both parts come naturally by strong induction on $i + j$. □

Hence, the maximal number of edits heuristic gives a lower bound of the Levenshtein distance.

3.3.6 Theoretical and Experimental Complexity

Like alignments, deciding the existence of multi-alignments is NP-complete for a size of runs bounded by an integer n given as input in unary (Section. 2.3.1). In this paper we propose to encode them as SAT instances and rely on efficient SAT solvers to compute the artefacts. The dominating factor in the time complexity of our technique is to solve the formulas, i.e. the call to the SAT solver dominates the complexity. The size of our formulas (and the computation time to construct them) are polynomial in the input.

More precisely, the formula that encodes the edit distance between two words of lengths n and m, has size $O((n+m) \times n \times m)$. The formula for runs of length n of a model $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ has size $O(|P| + n \times (|T|^2 + |F|))$. The $n \times |T|^2$ comes from the SAT formula (2.6) which enumerates pairs of transitions; it is immediate to encode the same constraint as a pseudo-SAT formula of size $O(n \times |T|)$ using the ability of pseudo-SAT to express directly constraints on the number of variables assigned to `true` among a set of variables. Hence, the encoding of the model runs has size $O(n \times |F|)$.

For multi-alignments, we need to repeat the encoding of the edit distance for each log trace. The size of the final formula sums to $O((n+m) \times n \times m \times |L| + n \times |F|)$, where m is the maximum length of log traces. The $(n+m)$ factor can be reduced significantly by setting a limit threshold to the value of d in the computation of edit distances when the distances of interest are expected to be sufficiently small. With this threshold, our formulas are essentially:

- linear in the size of the model
- linear in the size of the log
- quadratic in the length n of the considered anti- or multi-alignments. Actually, going further with the heuristic using threshold LIM for edit distance, one could eliminate all the variables $\delta_{i,j,d}$ with $|j - i| \geq \text{LIM}$ and then make our formulas linear also in n. We have not implemented this optimization.

The optimizations presented in Section 3.3.4 have a very significant impact in practice but do not change the theoretical complexity.

Technical Details

In practice, what limits our approach is mainly memory used by the solver. While in theory, solving a SAT formula requires only linear space, in practice, solvers tend to store information in order to improve their time complexity. On the other hand, the time required to solve our formulas in practice, does not grow as fast as the exponential than one could expect from the theoretical complexity.

3.3.7 Conclusion and Limit of the MinSAT Algorithm

In this subsection, we presented the first algorithm to compute multi-alignment with edit distance. Due to the complexity of the SAT formula that encodes the entire log sequences and the process model, the algorithm takes time and memory for getting the CNF form required by the solvers. Indeed, the translation of the SAT formulas to CNF creates a large number of variables [96]. In practice, the algorithm blows up for real-life instances. Moreover the SAT formula quickly appears as a black box which is hard to optimize and modify for non-experts. This is why we present in Section. 3.4 another algorithm that gets multi-alignments based on a very known base of algorithm, i.e., an A*-based algorithm.

3.4 An A* Algorithm for Computing Discounted Multi-Alignments

The A* algorithm is a very known search path algorithm for graph widely used for its completeness and optimality [65]. Like other works of the literature [4], we employ the reachability graphs of Petri nets to transverse process models with this algorithm. The novelty of our approach is the introduction of the *discounted edit distance* that prioritizes prefixes of alignments thus enabling the computation of more complex problems like multi-alignments.

The idea behind the discounted edit distance is motivated by the following use case: for certain processes, the costs associated to deviations at early stages of the process' execution are more important than the ones at the end. For instance, consider a loan application process that has two decisions: one at the beginning, assessing the type of customer (gold, silver, normal), and one at the end, determining whereas the loan was received in a labor day or not. It is normal that the stage in which these decisions are made in any possible execution of the process reveal their importance. For instance, if for the company it is very important to know the type of the customer because further information needs to be gathered depending on the customer's type, then it is likely that the corresponding process has the type of customer decision close to the start of any possible execution. On the contrary, if the day when the loan was received is not so important, then it is likely that the corresponding events will be pushed to the end of the traces.

The aforementioned situation holds for *knock-out processes* [123], representing processes where two outcomes are possible: OK or NOK. In these processes, ordering checks are usually observed, because it allows faster process executions. Indeed, as many tasks of those processes aim at determining the final output, the knock-out decisions should be taken at the beginning of the process, in growing order [99].

3.4.1 The Discounted Edit Distance

The idea of the *discounted edit distance* is to penalize more insertions and deletions when they occur at the beginning of the sequences, and less when they occur later.

Definition 16 (Discounted Edit Distance)**.** *We define the* Discounted Edit Distance *between two sequences u and v with discount parameter $\theta \geq 1$ by $\mathcal{D}_\theta(u,v) \stackrel{\text{def}}{=} \mathcal{D}^0_\theta(u,v)$ where:*

$$\mathcal{D}^k_\theta(u,v) = \begin{cases} \mathcal{D}^k_\theta(\langle\rangle,\langle\rangle) = 0 \\ \mathcal{D}^k_\theta(\langle\rangle, b.v') = \mathcal{D}^{k+1}_\theta(\langle\rangle, v') + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u',\langle\rangle) = \mathcal{D}^{k+1}_\theta(u',\langle\rangle) + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u', b.v') = \mathcal{D}^{k+2}_\theta(u', v') \qquad \textit{if } (a == b) \\ \mathcal{D}^k_\theta(a.u', b.v') = \min \begin{cases} \mathcal{D}^{k+1}_\theta(u', v) + \theta^{-k} \\ \mathcal{D}^{k+1}_\theta(u, v') + \theta^{-k} \end{cases} \quad \textit{otherwise.} \end{cases} \tag{3.18}$$

Hence, insertions and deletions cost θ^{-k} where k refers to the position where they occur.

Lemma 3. *For $\theta = 1$, the discounted edit distance corresponds to the Levenshtein distance.*

Proof. With $\theta = 1$, we have $\theta^{-k} = 1$ for any k and we obtain Def. 30 from Def. 34. □

Example 3.4.1 (Discounted Edit Distance)**.** *Let $u = \langle x, a, b\rangle$ and $v = \langle a, y, b\rangle$. The discounted edit distance between u and v is $\mathcal{D}^{u,v}_\theta = \theta^{-1} + \theta^{-4}$. If $\theta = 1$, the distance equals to 2 and is the Levenshtein edit distance where deleting x costs 1 and adding y costs 1.*

Technical Details

The Discounted Edit Distance is implemented in pm4py, in the boltmaud/pm4py branch, with a dynamic programming.

In practice, relevant values for the discount parameter θ are slightly larger than 1. For $\theta = 2$, the discount is already very severe since an edit at position k costs more than the sum of all the following edits.

Lemma 4. *With the Discounted Edit Distance, for $\theta \geq 2$, an edit at position k costs more than the sum of all the following edits.*

Proof. For u and v, two words, let k be the position of a non-free cost in $\mathcal{D}_\theta(u,v)$. We note its cost $c(k) = \theta^{-k}$.

The next edits can occur at positions $j \in \{k+1, \ldots, n\}$ where, in the worst case, $n = |u| + |v|$. We write $S(j,n)$ the sum of costs. The maximal value of this sum appears when only non-free edits are used by the discounted edit distance:

$$S(k,n) = \sum_{j=k+1}^{n} c(j) = \theta^{-(k+1)} + \theta^{-(k+2)} + \cdots + \theta^{-n} = \frac{\theta^{-k} - \theta^{-n}}{\theta - 1} \tag{3.19}$$

Hence, $c(k) = \theta^{-k} > S(k, n)$ for $\theta \in [2, \infty[$. Otherwise, in the best case, there is no edit after position k and the cost of the edit at position k is higher than a null sum. □

Observe that the discounted edit distance is not a proper distance metric because it does not have the triangle inequality property recalled below:

$$dist(u, w) \leq dist(u, v) + dist(v, w) \tag{3.20}$$

However, in this thesis, we abuse of the term *distance* for its resemblance to the classical edit distance. Indeed, by introducing this novel similarity metric, we aim at proposing a variant of the Levenshtein edit distance to approximate alignments. Other distances, like the cosine distance, also abuse of this term for similar reasons [97]. The discounted edit distance is a *semimetric* because it verifies the first axioms for a metric: $d(x, y)$ 0, $d(x, y) = 0$ *if and only if* $x = y$ and $d(x, y) = d(y, x)$. We give below a counter example of the triangle inequality property for the discounted edit distance.

Example 3.4.2 (Discounted Edit Distance does not verify the triangle inequality property)**.** *Let* $u = \langle b, b, a \rangle$, $v = \langle a, b \rangle$ *and* $w = \langle a, a \rangle$. *For* $\theta = 2.000000000000001$, *we have* $\mathcal{D}_\theta(u, w) = 1.5624999999999996$ *and* $\mathcal{D}_\theta(u, w) + \mathcal{D}_\theta(u, w) = 1.5624999999999991$ *which does not verify the triangle inequality property.*

3.4.2 Using the Discounted Edit Distance for Alignments

Similarly to the Levenshtein edit distance, the discounted edit distance can be applied to alignments. For $\theta = 1$, alignments based on the discounted edit distance is equivalent to classical alignment. However when $\theta > 1$, the costs are dynamic and depend on the number of previous edits.

Lemma 5. *For* $\theta > 1$, *an edit* ω *of position* j, *any move of position* $k > j$ *costs less than* ω.

Proof. Any function $f : k \to \theta^{-k}$ where $\theta > 1$ is strictly decreasing. Then for $j < k$, we have $\theta^{-j} > \theta^{-k}$. □

As a consequence, algorithms for computing optimal discounted alignments will tend to align in priority the prefixes of the log traces. Suffixes are less costly. From Lemma 4, when the discount parameter is $\theta = 2$, an edit of position j is more costly than the sum of all the next costs.

Example 3.4.3 (Prefixes First)**.** *Let* $\sigma = \langle s, f, a, g, d, d \rangle$ *a log sequence and the process model* N_2 *of Fig. 3.1a. The best classical alignment is* $\langle s, f, g, d, d \rangle$ *with a cost of 1 for the deletion of activity* a *in* σ. *When using the discounted edit distance with* $\theta = 2$, *the optimal alignment is* $\langle s, f, a, \tau \rangle$ *in order to align activity* a *and disabling an edit of cost* θ^{-5}.

3.4.3 A* Algorithm for Computing Discounted Alignments

To compute alignments by using the discounted cost function, we present an A*-based algorithm which assigns weights to the explored states according to the discounted cost function for alignment. To a state reached by a move ω occurring in position i, will be assigned the weight of its predecessor, increased by the cost

$$h(\omega, i, \theta) \stackrel{\text{def}}{=} (0 \text{ if } \omega \text{ is a synchronous move}, \theta^{-i} \text{ otherwise}). \tag{3.21}$$

As a result of Lemma 5, this heuristic aims at aligning prefix first more than suffixes.

The function h, based on the discounted cost function, is easily incorporated in the state-of-the-art algorithms for computing alignments.

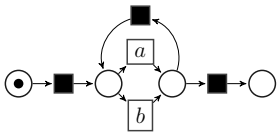
Synchronous Product Exploration

Our algorithm Alg. 1, noted $A^*SP_{\mathcal{D}=\theta}$, is inspired from [25]. It proceeds by exploring the state space of the synchronous product of the process model and the sequential Petri net representing the log trace as defined in Def. 12. An alignment corresponds to the shortest path between the initial marking to the final marking of the synchronous product. For this purpose, our A* algorithm maintains a priority queue Q of prefixes of runs, implemented as a heap of tuples $\langle \gamma, m, d \rangle$, for a prefix γ reaching marking m at cost d, such that the tuple with minimal cost d pops first. Line 1 initializes the heap with the empty prefix reaching the initial marking at cost 0, i.e. $\langle \langle\rangle, m_0, 0 \rangle$.

Algorithm 1: Computation of Discounted Alignments

Input : $SP = ((P, T, F, m_0, m_f, (\Sigma \cup \{\gg\})^2, \lambda))$: synchronous product, θ: discount parameter

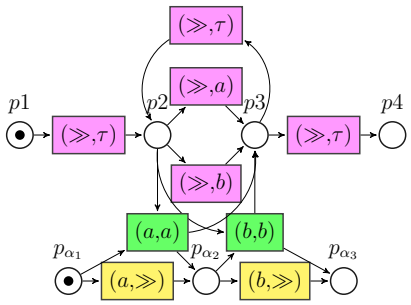
1 $Q \leftarrow \{\langle \langle\rangle, m_0, 0 \rangle\}$ // Heap of open states ordered by distance
2 $A \leftarrow \emptyset$ // Initialize closed set
3 **while** $Q \neq \emptyset$ // While not all reachable states visited
4 **do**
5 $\langle \gamma, m, d \rangle \leftarrow Q.pop()$ // Get next state minimizing d
6 **if** $m = m_f$ **then**
Return: $\langle d, \gamma \rangle$
7 $A \leftarrow A \cup \{\langle m, |\gamma| \rangle\}$ // Add state to closed set
8 **for** $t \in T$ *with* $m[t\rangle m'$ **do**
9 $\gamma' \leftarrow \gamma \bullet t$ // Get new prefix
10 **if** $\langle m', |\gamma'| \rangle \notin A$ // Reaching a not yet visited state
11 **then**
12 $d' \leftarrow d + h(t, |\gamma'|, \theta)$ // Compute cost of γ'
13 $Q \leftarrow Q.insert(\langle \gamma', m', d' \rangle)$ // Insert new prefix in heap
Raise : m_f is not reachable



(a) Process Model N_2

$\langle a, b \rangle$

(b) Log trace σ_2



(c) Synchronous Product between N_2 and σ_2.

Figure 3.2: Drawing the Role of Length in the States of the A* Algorithm.

Line 3 starts a *while* loop that ends only when the final marking is reached (line 6) or when the priority queue is empty (line 3). Line 9 gets the next firing transitions of the synchronous product. Some transitions correspond to the log and model moves and are costly. The other transitions are the synchronous moves and are free, like in the original algorithm.

Our discounted cost function h appears on line 12 and determines the cost of the new prefix. Line 13 adds the new discovered state in the priority queue with its prefix γ' and it cost d' for reaching m'.

When the algorithm reaches the final marking, line 6, the while loop is broken and the algorithm returns the sequence of firing transitions to reach the final marking. In fact, this sequence of transitions gives the sequence of moves of the alignment.

Role of Length in States

In the classical version of alignment computation, the state contains the markings only. However, the length of the current alignment plays an important role in the discounted cost function. Indeed, the first visit of a marking might not be the optimal one, as it is the case in the classical version of alignments. A same marking ω can be reached with different firing sequences of moves of different lengths. The first path that gives the first visit of the marking ω is the shortest one. Let's call this short path γ_{short}, and γ_{long} a longer path from the initial marking to this marking ω. We have $|\gamma_{long}| > |\gamma_{short}|$. However, γ_{short} might contain future edits to reach the final marking. If those edits are of position lower than $|\gamma_{long}|$, it questions the optimality of γ_{short}. We give an example of this situation below.

Example 3.4.4 (Role of Length in the Algorithm)**.** *Let's consider the simple process model N_2 on the left and the log sequence $\sigma_2 = \langle a, b \rangle$ of Fig. 3.2. Fig. 3.2c shows the synchronous product of N_2 and σ_2. Let's suppose that silent transition labeled by τ costs for this example. For the marking $\omega = \{p3 : 1, p_{\alpha_3} : 1\}$ of the synchronous product, two firing sequences compete for the minimization of the cost. Indeed, both $\gamma = \langle (\gg, \tau), (a, a), (\gg, \tau) \rangle$ and $\gamma' = \langle (\gg, \tau), (a, a), (\gg, \tau), (b, b) \rangle$ have a cost of $\theta^{-1} + \theta^{-3}$ and reach ω. However we notice that γ' has a synchronization at position 4, but we do not know yet what appear at position 4 for γ'. Then both paths should be kept.*

Notice that we tackled the problem of optimality of the alignment with the discounted cost function. For $\theta > 1$, this optimality does not correspond to the optimal classical alignment.

Process Model Exploration Along with Trace Exploration

In order to speed up the exploration, the alignment algorithm can simulate the synchronous product without explicitly constructing it. The synchronous product allows to easily play the moves of alignment. However, those moves can be found by exploring the process model and the trace separately. By comparing the next activity of the process model, given by the semantic of the net, and the next activity of the trace, we obtain the type of move. For instance, at the initialization, one possible next activity of N_2 of Fig. 3.2c is a and the first activity in σ is also a. Then, we can move forward with a synchronous move, like in the synchronous product but without constructing the corresponding transition of the move. Then the m in the algorithm (for the marking of the synchronous product) is replaced by a pair $\langle m, p \rangle$ where m is the marking of the process model and p the position in the trace. Any marking of $A^*SP_{\mathcal{D}=\theta}$ can be given into a couple $\langle m, p \rangle$ in this proposed simulation that we note $A^*PT_{\mathcal{D}=\theta}$. For instance, marking $\{p4 : 1, p_{\alpha_2} : 1\}$ of the synchronous product given in Fig. 3.2 corresponds to $\langle \{p2 : 1\}, 1 \rangle$ where $\{p2 : 1\}$ is the marking in N and 1 the position in σ. The final marking becomes $\langle m_f, |\sigma| \rangle$ where the trace has been read and the process model reaches its final places.

Technical Details

Both algorithms are implemented in a branch of pm4py. Parameter SYNCHRONOUS of *pm4py.algo.conformance.alignments.variants.dijkstra_exponential_heuristic* class allows choosing between the two versions.

3.4.4 Comparison to Classical Alignments

Due to the discount parameter θ in the discounted cost function, our heuristic prioritizes the alignment of the beginning of the log trace. In the algorithm, this difference with the

classical alignment algorithm appears line 5 of Alg. 1 where the markings that minimize the cost are much more different with the discounted cost function than by using the classical cost function for alignments. Indeed, when costs are all equivalent, many paths compete in the search for the optimal alignment. However, with very different costs, the number of paths with similar costs is low, thus reducing the search space. This characteristic is well observed in practice with a reduction of runtime.

Example 3.4.5 (Reducing the Search Space with the Discounted Parameter)**.** *For the example of Fig. 3.1a, there is a first choice between f, c, b and g. For large θ, the decision is quickly given thus disabling testing the depth of the other paths. Let $\sigma'_2 = \langle s, f, b, a\rangle$ the first trace of L_2. With $\theta = 2$, $\langle s, f\rangle$ is already the optimal prefix because f allows a synchronizations where the other paths give a cost of θ^{-3} for a model move.*

3.4.5 Heuristic for Reducing the Search Space of Alignment Computation

The search space of $A^*SP_{\mathcal{D}=\theta}$ is large and even larger than the Dijsktra-based algorithm for alignments due to the incorporation of the lengths of the runs that reach the same marking. To reduce it, we come back to the classical closed set that contain the markings only. Every $\langle m, |\gamma|\rangle$ of the closed set A is reduced to m (like in [25]).

With this reduction, only the first paths that reaches the marking are used. When several concurrent firing sequences of equal cost exist, line 5 picks one as the optimal path and line 8 classifies the marking in A. Then the other firing sequences of equal cost for this marking are not considered anymore (line 10).

In practice, the loss of quality is very limited: we observed that the alignments found by the modified algorithm have very similar discounted cost than without this heuristic.

3.4.6 Adaptation of the A* Algorithm for Multi-alignment

Multi-alignment computation involves the entire log. Thus the previous algorithm which uses the construction of the synchronous product or its simulation does not work for multi-alignments. However, we present in Alg. 2 an adaptation of the algorithm where we transverse the reachability graph of the process model only. Each state is still a tuple $\langle \gamma, m, d\rangle$ where m is a marking and γ the prefix of run that reaches it for a distance d. Now, weights of states are given by the heuristic:

$$h(\gamma, L, \theta) \stackrel{\text{def}}{=} \max_{\sigma \in L} \left(\mathcal{D}_{\theta,}(\gamma, \sigma) - \frac{\theta^{-|\gamma|+1} - \theta^{-(|\sigma|+|\gamma|)}}{\theta - 1} \right) \tag{3.22}$$

which depicts the maximal distance of the prefix γ to the log traces by using the discounted edit distance and prevents, with $-\frac{\theta^{-|\gamma|+1}-\theta^{-(|\sigma|+|\gamma|)}}{\theta-1}$, the best alignment suffix of u for each

$\sigma \in L$. The minimization of this maximal distance to the log traces is found by the search of the shortest path. Once the final marking is reached, the exact maximal discounted edit distance $\max_{\sigma \in L} \mathcal{D}_{\theta,}(\gamma, \sigma)$ of the found u to the log is computed (line 8). The algorithm checks that no other run beats this best multi-alignment before stopping.

This algorithm involves the computation of the distance for each trace at each step which is the weakness of the methods. However, the algorithm is able to deal with large instances as the memory space of the algorithm is polynomial.

Algorithm 2: Computation of Multi-Alignment by using the Discounted Distance

Input : $N = (P, T, F, \lambda, m_0, m_f)$: process model,
L: log,
θ: discount parameter

1 $Q \leftarrow \{\langle \langle\rangle, m_0, h(\langle\rangle, L)\rangle\}$ // Heap of open states ordered by distance
2 $\mathcal{B}_\gamma \leftarrow$ undefined // Current best multi-alignment
3 $\mathcal{B}_\delta \leftarrow -\infty$ // Current best distance to reach m_f
4 **while** $Q \neq \emptyset$ // While not all states visited
5 **do**
6 $\quad \langle \gamma, m, d \rangle \leftarrow Q.pop()$ // Next state maximizing d
7 $\quad$ **if** $m == m_f$ **then**
8 $\quad\quad \delta \leftarrow \max_{\sigma \in L} \mathcal{D}_\theta(\gamma, \sigma)$ // Exact distance δ
9 $\quad\quad$ **if** $\mathcal{B}_\delta > \delta$ **then**
10 $\quad\quad\quad \mathcal{B}_\gamma \leftarrow \gamma$ // New best multi-alignment
11 $\quad\quad\quad \mathcal{B}_\delta \leftarrow \delta$ // Update distance
12 $\quad$ **for** $t \in T$ *with* $m[t\rangle m'$ **do**
13 $\quad\quad \gamma' \leftarrow \gamma \cdot t$ // Get new prefix
14 $\quad\quad d' \leftarrow h(\gamma', L)$ // Get *possible* distance of γ' to L
15 $\quad\quad Q \leftarrow Q.insert(\langle \gamma', m', d' \rangle)$ // Place new state

Output: $\mathcal{B}_\gamma$: best multi-alignment,
$\mathcal{B}_\delta$: minimal distance of $\mathcal{B}_\gamma$ to L

Preempting the Best Suffix

Here, we explain the need of the fraction $-\frac{\theta^{-|\gamma|+1}-\theta^{-|\sigma|+|\gamma|}}{\theta-1}$ of the heuristic function.

The heuristic preempts an approximation of the best suffix γ' for γ. In the best case, where $\max_{\sigma \in L} \mathcal{D}_\theta(\gamma \bullet \gamma', \sigma) = 0$, either $\gamma' = \langle\rangle$ and $\max_{\sigma \in L} \mathcal{D}_\theta(\gamma, \sigma) = 0$, either edits between γ and σ are log moves and synchronizations with γ' appear after position $|\gamma|$ and, in the worst case, for all activities in σ. The sum of costs $S(|\gamma| + 1, |sigma| + |\gamma|) = -\frac{\theta^{-|\gamma|+1}-\theta^{-(|\sigma|+|\gamma|)}}{\theta-1}$ (c.f. Proof of Lemma. 5 on page 40) preempts such situation by supposing that every activity of σ aligns with γ' from position $|\gamma|$. The worst case is when $\max_{\sigma \in L} \mathcal{D}_\theta(\gamma \bullet \gamma', \sigma) = |\gamma| + |\gamma'| + |\sigma|$. Fortunately the fraction $-\frac{\theta^{-|\gamma|+1}-\theta^{-(|\sigma|+|\gamma|)}}{\theta-1}$ alleviates

potential future alignments of every activity contained in σ.

The priority queue stores the prefixes of multi-alignments. Like for alignments, this queue gives an approximation because the discounted edit distance does not verify the triangle inequality. Equivalently, we observe good results in practice.

Example 3.4.6 (Suffix Multi-alignment)**.** *Let consider model* N_1 *of Fig. 3.1a, log* $L1' = \{\langle s, f, b, a\rangle, \langle s, c, b, a\rangle, \langle s, c, b\rangle\}$ *and* $\theta = 2$*. For the prefix multi-alignments* $\gamma = \langle s, c, b\rangle$*, the heuristic function* h *is* $h(\gamma, L1', 2) = 2^{-3} + 2^{-4} + 2^{-7} - \frac{2^{-4}-2^{-7}}{1} = 0.14$ *where the further log trace is* $\sigma = \langle s, f, b, a\rangle$*. The fraction preempts potential future synchronizations which can appear on position* 4 *in the best case. The prefix* $\gamma' = \langle s, g, d\rangle$ *costs* $h(\gamma, L1', 2) = 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} - \frac{2^{-4}-2^{-7}}{1} = 0.19$.

For models with loops, the *while* loop continues infinitely. To break it, we use a limit of reaching the final marking. We also applied this same heuristic to any marking which brings us the next nice heuristic.

3.4.7 Heuristic for Reducing the Search Space of the A*-based Algorithm for Computing Multi-alignment

When models are dense in choices, concurrency and loops, some markings are visited a large number of times. For instance, all possibles combinations of a concurrent part finally reach the same marking. Because best prefixes are prioritized, a simple but efficient heuristic is to limit the number of times a marking can be reached. We propose a parameter μ for this purpose and observe good results in practice.

3.4.8 Conclusion of the A*-based Algorithm

Thanks to our discounted edit distance, we developed an A*-based algorithm which reduces the search space for alignment by disabling runs that do not align well at the beginning of the sequences. The algorithm is adapted to multi-alignment computation which allows to obtain model-based trace variants for real-life logs which was not possible with the SAT-based algorithm. We present concrete experiments in the next section.

3.5 Experiments

The algorithms presented in this chapter have been implemented in the three tool presented in Section.1.4.4 : Darksider, da4py and pm4py. The MinSAT-based method is given in both Darksider and da4py. The former is out-of-date for multi-alignments because it contains only the minimization of the sum of distances while the preferred goal is the minimization of the maximal distance. Moreover, the latter gives the fastest results because it provides the optimized version of the SAT encoding and allows setting state-of-the-art SAT solvers.

The A*-based algorithms are developed in a branch of pm4py in the aim of providing a large range of access to multi-alignment computation. This algorithm gives fast approximations that allows to obtain the artefact for real-life instances. Some tutorials for launching alignments and multi-alignments by using the three tools are given in Appendices. A.1 and A.2.

This section presents a set of experiments of the different algorithms for both alignment and multi-alignment computation. Indeed, despite this chapter focuses on multi-alignments, the contribution brings some novel algorithms for computing alignments too. We compare their quality and runtime to the state-of-the-art methods. About multi-alignments, we first show the improvement in term of CNF clauses, obtained with our formula optimization. Then we give a comparison between our SAT algorithm and the A*-based approach. We provide some runtimes to position the search of this artefact. Finally, we give a case study where we figure how interesting multi-alignments can represent a collection of log sequences.

For some experiments, we give the associated scripts and outputs in Annexe. B.

3.5.1 Comparing our Discounted Alignments with the State-of-the-art Metholds

This section aims at showing a comparison of quality and runtimes between the proposed versions of alignments and existing ones. In this section we only tackle the A*-based algorithms because the SAT-based version does not handle large inputs. We first present general comparisons of the alignments methods and stress on the impacts of the discounted parameter. Then, we zoom on particular cases and add other methods similar to our.

Comparison with respect to baselines and Influence of the Discounted Parameter θ

We experimented the algorithms for both artificial and real-life logs and the corresponding models. Artificial set is taken from [112] and contains large models. For real-life logs, we used data given in the Business Process Intelligence Challenges from 2012 to 2020. We mined the process models of those logs with methods of the literature[1]. First, we applied a preprocessing method[2] introduced by [57] to extract good prototypes of the logs. This preprossessing step allows us to obtain precise but not perfectly fitting models when using miners, interesting for alignment comparison. Then, we launched two discovery algorithms on the found prototypes: the inductive miner [75] and the split miner 2.0 [10]. As the latter tool gives BPMN models, we transform them into Petri nets with ProM software [130].

We computed the alignments on *variants* only, i.e., unique sequences of activities. This choice of using variants only allows to correctly compare the method's runtimes in case

[1]available at https://github.com/BoltMaud/A-Discounted-Cost-Function-for-Fast-Alignments-of-Business-Processes-Sources

[2]Prototype Selection plugin of Prom software with default settings

Log	#variants	$\|\Sigma\|$	$\max_{\sigma \in Log} len(\sigma)$	Model	$\|T\|$	$\|P\|$	$\|F\|$
L1	453	36	37	M1	39	40	92
L2	500	32	52	M2	34	34	80
L3	462	109	217	M3	123	108	276
L4	496	44	176	M4	52	36	106
L5	500	32	71	M5	33	35	78

(a) Artificial Logs and Models

Log	#variants	$\|\Sigma\|$	$\max_{\sigma \in Log} len(\sigma)$	Model Miner	$\|T\|$	$\|P\|$	$\|F\|$
BPI2012	4366	24	175	IM	34	24	68
				SM	30	23	60
$BPI2018_{pa}$	3832	24	100	IM	22	24	60
				SM	20	15	40
BPI2019	11973	42	990	IM	18	13	38
				SM	13	10	26
$BPI2020_{dd}$	99	17	24	IM	15	11	32
				SM	14	9	28
$BPI2020_{rp}$	89	19	20	IM	31	26	74
				SM	23	12	46

(b) Real-life Logs and Models

Figure 3.3: Input Description for Alignment and Multi-alignment Experiments

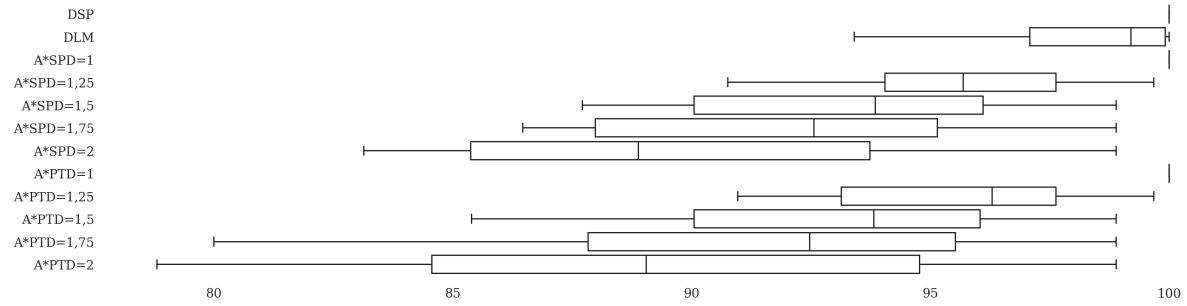
that one reduce the log to variants and not another one. Appendix. B.1 provides the script of those experiments and Tab. 3.3 gives an overview of the inputs.

We compare our alignment results to the four current methods of the state-of-the-art implemented in pm4py which are: -the Dijkstra search on the Synchronous Product without heuristic (DSP)[4], -a Disjkstra that consumes less memory by using a similar idea of our second algorithm (DLM), -an A*-based algorithm on the state-space of the synchronous product that incorporates an heuristic on reaching the final marking (A^*SP_{mf}) [132] and -its less-memory version (A^*LM_{mf}). We recall notation of our methods $A^*SP_{\mathcal{D}=\theta}$, for the version that uses the synchronous product, and $A^*PT_{\mathcal{D}=\theta}$, for the second one that explores only the process model and the trace.

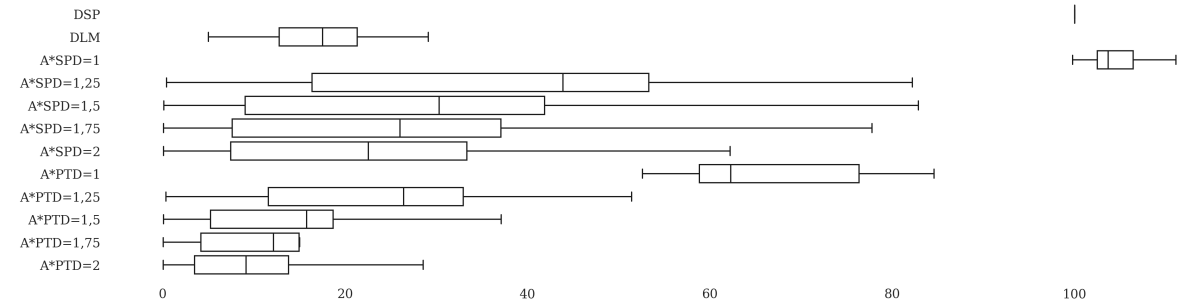
Results. The *quality* of an alignment found by a heuristic method, is defined as the ratio (in %) between the classical cost (number of model or log moves) of the optimal alignment (given by the DSP method) and the classical cost of the alignment found by the method. In Fig.3.4a we give the quality of each method.

Similarly, in Fig.3.4b, each line shows the sum of the runtimes of alignment computations by a method, expressed in percentage of the runtime of the DSP method. The runtime reflects the space of search. The box charts have wide range because they summarize the results of all the alignments which are very different (depending on both the model and the log involved). We see that the runtime of the DLM algorithm is 20% of the runtime of the DSP method. For our heuristic $A^*PTD = 2$, the average runtime is around 10% of the reference method DSP (which corresponds to a gain of 90% of runtime, the result of a large reduction of the search space), for an average quality between 90 and 85% of the optimal alignments.

We did not represent in the charts the runtimes for methods A^*SP_{mf} and A^*LM_{mf} (implemented in pm4py) since they are much higher than the others: A^*SP_{mf} ran up to 30 times longer than the DSP and A^*LM_{mf} up to 7 times longer.



(a) Quality of the alignments obtained by different methods. Quality is defined as the ratio (in %) between the classical cost (number of model or log moves) of the optimal alignment and the classical cost of the alignment found by the method. The first line is the baseline and present the optimal approach which has no loss in quality, i.e., we observe 100% of quality. In general, one can see that the loss of quality with our heuristics is rather limited.



(b) Runtime (in % of the runtime of the DSP Algorithm). A percentage lower than 100 corresponds to a gain of runtime compared to the DSP method.

Figure 3.4: Comparison of Quality and Runtime of Different Methods for Computing Alignments.

Experiment Conclusion

Compared to methods given in the Python library pm4py, we observe that our method provides a better compromise between runtime and quality of alignments. Moreover, these experiments aim at showing the influence of the discount parameter θ on quality and the search of space. The quality decreases when θ raises. However, the gain in term of search is high for $\theta > 1$.
We observe that the two charts of Fig. 3.4a and 3.4b are correlated. In other words, the small lost of quality is correlated with the gain of runtime and a compromise between the two dimensions can be set with the parameter θ.

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Async. Moves	14	14	14	14	14	–	14	14	14	15	15	14	14	14	15	15
Runtime (s)	0.14	0.62	0.73	0.02	0.01	–	0.10	0.10	0.04	0.04	0.02	0.13	0.12	0.06	0.04	0.04

(a) BPI2020$_{rp}$

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Async. Moves	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
Runtime (s)	47.78	628.75	3385.59	1.55	0.06	1.38	13.28	0.02	0.01	0.01	0.01	43.41	0.03	0.02	0.02	0.01

(b) M5

Method	DSP	DLM	A^*SP_{mf}	A^*LM_{mf}	PNR	REC_{ilp}	A^*SPD					A^*PTD				
							1	1.25	1.5	1.75	2	1	1.25	1.5	1.75	2
Async. Moves	36	38	36	36	36	–	36	886	2137	1898	4265	36	161	146	330	338
Runtime (s)	6.19	2121.98	16011.60	0.76	1.94	–	4.38	0.07	0.14	0.13	0.45	8.24	0.10	0.12	0.07	0.12

(c) M3

Figure 3.5: Particular Alignments that Draws Advantages and Disadvantages of our Method. Run on a MacBook air 2017 model with a 1.8 GHz Intel ® Core^{TM} i5 CPU and 8G RAM.

Zoom on Particular Alignments

The omission of ProM and other tool results in the previous section is due to the differences between the output formats which made difficult the comparison of quality and runtime. However, in this section we zoom in particular cases, i.e., by running a log sequence only, thus making human interpretation possible. We add PNR the results given by the PNetReplayer package of [7] in ProM and REC_{ILP} the results given by [111].

We choose 3 models and traces that have particular characteristics. First, we run the alignment between the first trace of BPI2020$_{rp}$ log and model IM because this couple (BPI2020$_{rp}$, IM) gives the least differences between the methods (Fig. 3.5a). The model has only 2 parallelism patterns and no loop. Then, we run the alignment of the first trace of $L5$ and model $M5$ where our method specifically works well (Fig. 3.5b). This model contains a concurrent pattern including 28 transitions and one loop. Finally, we present an alignment of the fifth trace of L3 which is very long (215 activities) and its model which has many loops and many parallelism patterns (Fig. 3.5c). The latter aims at showing the weakness of our method.

Results. From the three tables of Fig. 3.5, we observe that the methods usually find exact alignments. This is not true for the last experiment given in Fig. 3.5c which highlights our weakness. This due to the size of the trace (215) that, for the high base of logarithm θ, the algorithms face a situation where all the markings have the same cost (θ^n where n is very large borders zero). At this point, we already advise the user to check the length of the traces to set the discounted parameter θ (or to tackle very small differences between costs with an implementation where more decimal are allowed).

Observe now that for Tab. 3.5a and 3.5b using high value of θ brings very fast result for nearly optimal alignment in Tab. 3.5a and optimal alignment in Tab. 3.5b. Moreover, this

latter result even beats all the other methods including the ProM implementation (noted PNR). The particularity of model M5 is the large concurrency pattern that creates many paths of different behaviors. Most methods have to explore the different combinations created by the concurrency pattern. Our discounted function favors only the path that align at the beginning of this pattern and does not consider the other combinations of the activities.

The versions using less memory seem to be much less efficient sometimes even for less quality (see DLM for $M3$). The method REC_{ilp} worked only for the second model. $M3$ is too large for the Gurobi open source version and format of model 2020_{rp} is not accepted by the tool.

Experiment Conclusion

These experiments depict situations where our methods beat the state-of-the-art algorithms for alignments. However, some cases are still to improve when the parameter θ is high due to the limit of computer's float numbers which are shortened.

Comparison with the Token Replay Approach

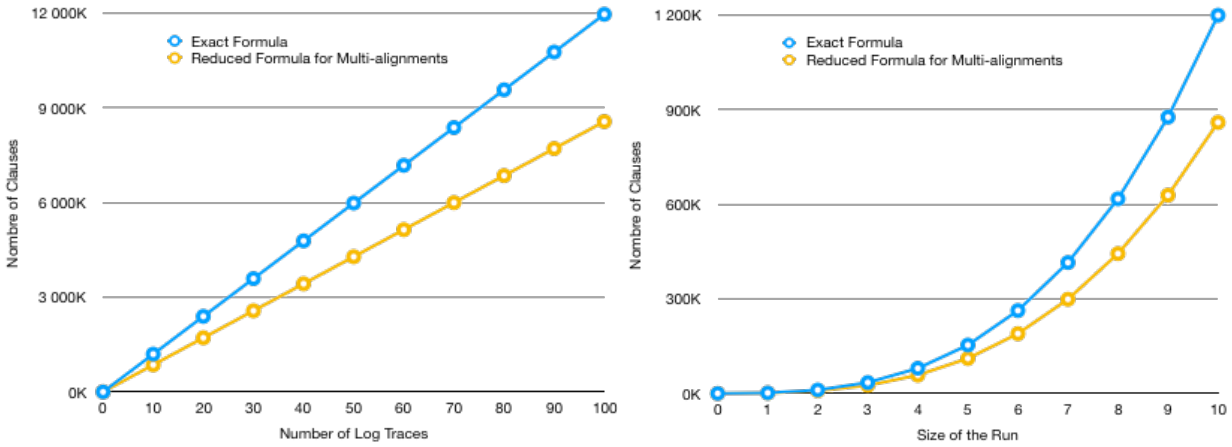
Last but not least, we give a comparison of runtimes between our algorithms for computing alignment and the token replay approach given in [14] because it also computes an approximation of conformance (more precisely fitness) of process models for a given log. For this experiment, we set our method with $\theta = 2$. We observe in Tab. 3.1 that our second algorithm gives faster results in most times. The token replay is however much faster for BPI2018_{pa}. We plan to compare fitness approximation in future work.

Method	BPI2012		BPI2018_{pa}		BPI2019		BPI2020_{dd}		BPI2020_{rp}	
	IM	SM	IM	SM	IM	SM	IM	SM	IM	SM
A^*SP	78.18	69.39	493.96	43.92	143.25	103.99	0.43	0.23	1.08	0.24
A^*PT	25.03	19.71	419.37	11.15	42.14	26.61	0.14	0.09	0.70	0.09
Token replay	35.41	36.86	36.11	31.01	45.99	49.40	0.20	0.19	0.22	0.18

Table 3.1: Runtime Comparison (in seconds) for Computing Discounted Alignments and the Token Replay Method Given in [14], Run on an on a MacBook air 2017 Model with a 1.8 GHz Intel ® Core™ i5 CPU and 8G RAM.

3.5.2 Computing Multi-alignment

This section provides multi-alignment experiments in different perspectives. We first show the impact of the SAT formula optimization.



(a) Number of clauses per formula for a size of run to 10 by increasing number of traces

(b) Number of clauses per formula for a log of 10 traces by increasing size of run

Figure 3.6: Comparison of Number of CNF Clauses of Produced Formulas for a Model of 10 Transitions.

SAT Formulas Reduction

In Section 3.3.4, we show a reduction of the SAT encoding. To convince readers, we present Fig. 3.6 that reports the size of the formulas in term of CNF clauses. The formulas have been created for a proces model of Fig. 3.1a and some traces of the same alphabet. In Fig. 3.6a, for 0 traces, the number of clauses of the formula represents the number of clauses needed to create the SAT encoding of the model, described in Section 2.3.2. This number is not significant when compared to the size of the edit distance formulas as the number of traces increases. In Fig. 3.6b, we show the number of clauses when increasing the size of the run. As expected, the exact formula creates much more clauses than the optimized variant.

Experiment Conclusion

Thanks to our SAT formula reduction for multi-alignment, we considerably reduce the number of clauses contained the edit distance encoding which is a great improvement of this method whose weakness is its large memory use.

Comparison of Algorithms for Multi-alignment Computation

In this section, we give several comparison between the SAT-based implementation and the A*-based algorithm for computing multi-alignments.

First, we present some experiments for a small log of the literature, given in Fig. 3.7, and its associated process models where the SAT-based method can fully be computed. We did not consider all the process model usually treated with this log because some are

redundant in the search of multi-alignments (like several loops on activities which never occurs in the log, then one example with loop is sufficient) or not necessary (like the "Most Frequent Trace" where there is only one run). The chosen models are shown in Fig. 3.8 to Fig. 3.13.

The parameter "size_of_run" of the MinSAT-based algorithm is set to 11 which is large enough to get optimum multi-alignments to all the proposed models and the maximal number of edits is set to 22 which is optimal for this size of the run. For lower values, these parameters would compute the heuristics presented in Section. 3.3.5.

We set the heuristic μ of the A*-algorithm to 200 which limits the number of times we reach the same markings. We run the experiments for $\theta = 1.01$ and $\theta = 2$ to show the advantages and disadvantages of the discounted parameters.

$$
\begin{aligned}
&[\langle A, B, D, E, I\rangle,\\
&\langle A, C, D, G, H, F, I\rangle,\\
&\langle A, C, G, D, H, F, I\rangle,\\
&\langle A, C, H, D, F, I\rangle,\\
&\langle A, C, D, H, F, I\rangle]
\end{aligned}
$$

Figure 3.7: Small Artificial Log L_a for Experiments.

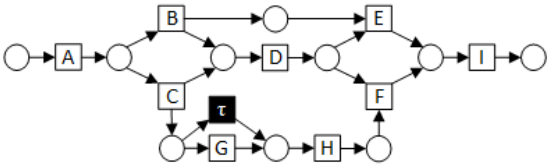


Figure 3.8: Generative model.

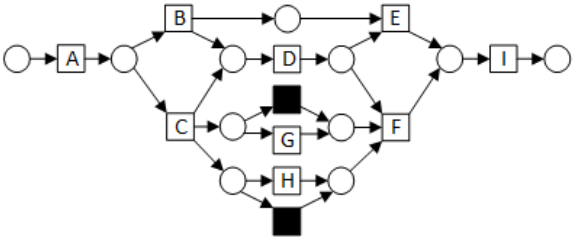


Figure 3.9: Model with G and H in parallel.

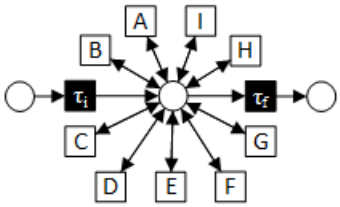


Figure 3.10: The flower model.

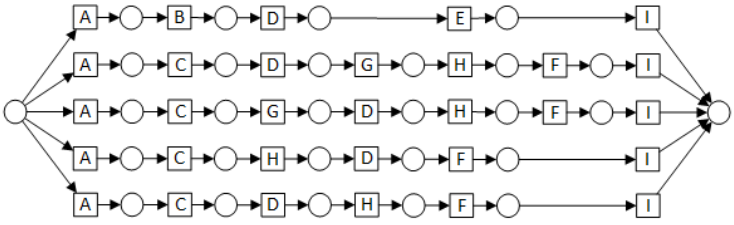


Figure 3.11: All traces separate.

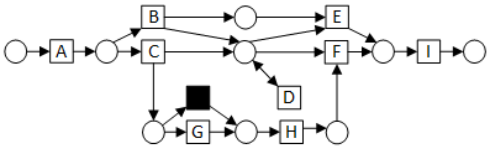


Figure 3.12: Model with D in a self-loop.

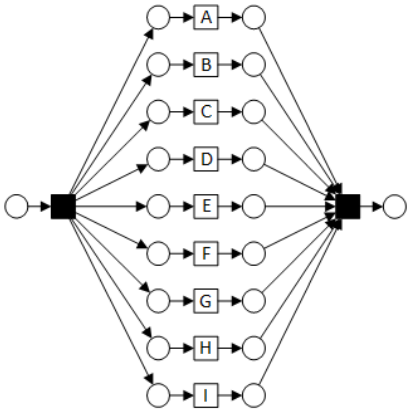


Figure 3.13: Model with all transitions in parallel.

Results. Tab. 3.2 shows the results. Column "Maximal distance" is the maximal Levenshtein edit distance between the discovered multi-alignments and the log traces. We observe that for $\theta = 1.01$, we always get an optimal multi-alignment, i.e., the maximal distance equals the optimum given by the SAT-based algorithm. Sometimes the multi-alignments are different for the same model but we recall that several optimum can exist for a given input. The use of the heuristic μ has no impact on the quality of those examples and gives fast results. For larger values of μ, the runtime considerably raises. For instance for $\mu = 2000$, the flower model takes 174.27 secs. In practice, this parameter can even be set to very low value, like 5, and give the optimum solution.

For $\theta = 2$, we do not get optimum multi-alignments for two models, i.e, the flower model and the all parallel model. However, we observe that the difference of quality is not bad and the runtime is much improved. This improvement is due to the limitation of the search space obtained with the parameter θ. As shown in the alignment's experiments, this parameter helps one to obtain a compromise between fast and efficient results. For the other models, the high value of the discounted cost does not affect the results, i.e., aligning prefixes first work well for those small artificial examples which questions the application on larger instances.

Experiment Conclusion

For a small and quite homogeneous log and its associated process models, we obtained very similar multi-alignment by using the SAT-based algorithm and the A* based algorithm. We showed that the discounted parameter helps to reduce in search space with a gain in runtime for a small lost of quality. Overall, in those experiments, the discovered multi-alignments give model-based representations of the log traces. However, this artificial inputs are very restrictive.

We extend our experimentation to larger logs and real-life instances. We used the models previously presented in Section. 3.5.1 for experimenting alignments. Memory issues

Model	Algorithm	Multi-alignment	Maximal Distance	Time (s)
Generative model (Fig. 3.8)	SAT	$\langle A, C, D, tau, H, F, I\rangle$	5	17.79
	A* with $\theta = 1.01$	$\langle A, C, \tau, D, H, F, I\rangle$	5	0.02
	A* with $\theta = 2$	$\langle A, C, \tau, D, H, F, I\rangle$	5	0.01
G and H in parallel (Fig. 3.9)	SAT	$\langle A, C, D, \tau, \tau, F, I\rangle$	4	17.58
	A* with $\theta = 1.01$	$\langle A, C, \tau, \tau, D, F, I\rangle$	4	0.06
	A* with $\theta = 2$	$\langle A, C, \tau, \tau, D, F, I\rangle$	4	0.02
Flower model (Fig. 3.10)	SAT	$\langle \tau, A, B, C, DF, I, \tau\rangle$	4	19.59
	A* with $\theta = 1.01$	$\langle \tau, A, D, F, I, \tau\rangle$	4	13.55
	A* with $\theta = 2$	$\langle \tau, A, \tau\rangle$	7	5.01
Separate Traces (Fig. 3.11)	SAT	$\langle A, C, D, H, F, I\rangle$	5	43.86
	A* with $\theta = 1.01$	$\langle A, C, D, H, F, I\rangle$	5	0.01
	A* with $\theta = 2$	$\langle A, C, D, H, F, I\rangle$	5	0.01
D as self-loop (Fig. 3.12)	SAT	$\langle A, C, D, \tau, H, F, I\rangle$	5	17.07
	A* with $\theta = 1.01$	$\langle A, C, \tau, D, H, F, I\rangle$	5	0.53
	A* with $\theta = 2$	$\langle A, C, \tau, D, H, F, I\rangle$	5	0.09
All parallel (Fig. 3.13)	SAT	$\langle \tau, A, B, C, D, G, H, E, F, I, \tau\rangle$	6	21.46
	A* with $\theta = 1.01$	$\langle \tau, A, C, B, D, G, H, F, E, I, \tau\rangle$	6	159.76
	A* with $\theta = 2$	$\langle \tau, A, C, B, D, E, I, F, H, G, \tau\rangle$	9	0.7

Table 3.2: Comparison of Multi-alignments for the Small Artificial Log of Fig. 3.7 and Models of Fig. 3.8 to 3.13 Where the SAT-based Algorithm Can Be Fully Executed, Obtained on a Virtual Machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM. A*-algorithm is Set With $\mu = 200$.

of the MinSAT-based algorithm forced us to reduce the size of the logs. We ran the experimentation for the 50th first traces of each only. Moreover, we used the heuristics of the SAT encoding and set the size of the run to 10 and the associated optimal number of edits to 20. For the A* version, we set the discounted parameter θ to 1.5 which slightly reduces the search space and we keep μ to 200. The script of those experiments is given in Appendix. B.2.

Results. The results are given in Tab. 3.3. We observe that the maximal distance of the multi-alignment to the log traces is high for some inputs. This result is due to the heterogeneity that exists in real-life event logs. Log BPI2012 and BPI2019 especially have large maximal distances but those logs are very complex with 4 366 trace variants for BPI2012 and 11 973 trace variants for BPI2019. Moreover, some lengths of traces have up to 990 activities. In opposite, multi-alignments of logs BPI2020_{dd} and BPI2020_{rp} are much closer to the log traces and are much more in the insight of the artefact which is to represent a collection of log traces.

Overall, the two algorithms find similar maximal distance of multi-alignments. But we observe that in most cases, the A* algorithm finds a multi-alignment closer to the log traces than the SAT-based method, i.e., the quality of the multi-alignments found with the A* algorithm is better than the quality of the ones found with the SAT implementation.

Log	Model	Maximal Distance		Time (s)	
		SAT	A*	SAT	A*
L1	M1	29	24	347.33	42.81
L2	M2	58	50	298.54	337.18
L3	M3	–	–	–	–
L4	M4	129	128	439.91	8048.28
L5	M5	–	–	–	–

Log	Model	Maximal Distance		Time (s)	
		SAT	A*	SAT	A*
BPI2012	IM	104	130	297.15	14.95
	SM	107	103	270.57	2.17
$BPI2018_{pa}$	IM	42	39	207.86	2785.75
	SM	42	34	184.77	58.92
BPI2019	IM	250	239	167.94	256.69
	SM	250	225	128.78	89.90
$BPI2020_{dd}$	IM	9	8	151.72	0.04
	SM	9	7	140.38	0.08
$BPI2020_{rp}$	IM	10	8	283.64	2.43
	SM	7	8	216.93	0.08

Table 3.3: Multi-alignment Approximation for the Logs and Models given in Fig. 3.3 by using the SAT Algorithm Implemented in da4py with Parameters $size\_of\_run = 10$ and $\max_d = 20$ and the A* Algorithm Implemented in pm4py with Parameters $\theta = 1.5$ and $\mu = 200$, run on a virtual machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM.

This result is due to the use of the heuristics of the SAT-based algorithm which does not guarantee the optimum solution while we kept similar setting than in the previous experiments for the A* algorithm.

In term of runtimes, we observe, in most cases, a large difference between the two algorithm. The SAT-based algorithm runs slower than the A* algorithm. However, we note some particularities where the runtime of the A* algorithm is larger than the SAT-based algorithm. Those models contain much concurrency but we already explained this weakness of our method for this structure.

Experiment Conclusion

For large inputs, the SAT-based algorithm for computing multi-alignments is limited due to memory issues. Moreover the gain on runtime by using the A* algorithm is well highlighted. We observed some consequences on multi-alignment quality with the SAT approach due to the use of the heuristics, which convinces again the use of the A*algorithm. However, the more a process model contains concurrent structures the least the A* algorithm runs fast and in some cases, using the SAT-based algorithm is even better. Overall the quality of the discovered multi-alignments is similar. We note that we obtain bad representatives for complex logs.

3.5.3 Multi-alignments as model-based trace variants: a Case Study

We observed in Tab. 3.3 that the found multi-alignments are quite far from the log traces which is in contradiction with the aim of this conformance checking artefact. In this section, we show how one can discover good quality multi-alignments.

We use the Loan Application log [23] whose brief description is given in Tab. 3.4. The column "$\#variants$" gives the number of variants of traces, i.e., unique sequences of activities. 100 variants is too high for human analysis. By discovering multi-alignment, we claim to provide a model-based trace variant such that the maximal distance between them and the log traces is given. However, event log contain many different behaviors and getting a unique model-based trace variants is not accurate. To convince the reader, we compare multi-alignments for the entire log and for some sub-logs.

$\|L\|$	$\#variants$	$\|\Sigma\|$	$\max_{\sigma \in L} len(\sigma)$
500	100	31	37

Table 3.4: Loan Application Log Description

To do so, we first mine a process model by using the inductive miner. We obtain the process model of Fig. 3.14. The model contains many choices, a concurrent structure and a loop.

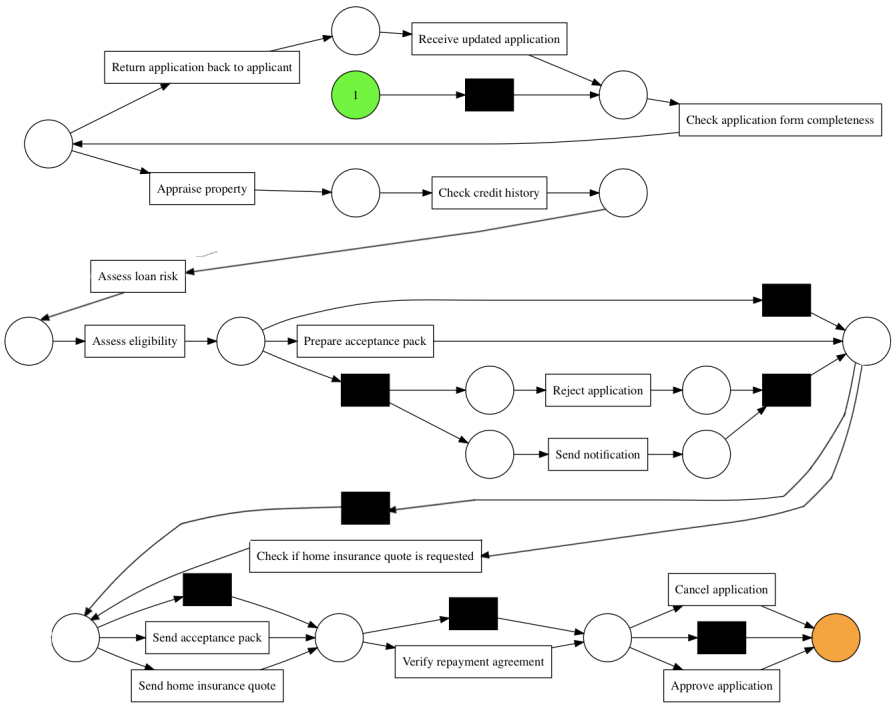


Figure 3.14: Process Model of the Loan Application Discovered with the Inductive Miner

Method	Maximal Distance of Multi-alignments to Log	
	max	avg
Entire Log	26	26
Random selection, sets of 10 sequences	26	20.9
Random selection, sets of 20 sequences	26	22.8
Kmean with 3 clusters	10	9.3
Kmean with 6 clusters	8	7

Table 3.5: Experimenting Multi-alignments for Different Sets of Log Sequences

Multi-alignments for the entire Loan Application log and the discovered process model have a maximal Levenshtein distance to the log sequences of 26. In other words, a multi-alignment of this model for the entire log sequences is a run such that any log sequences can be aligned to it for a maximum of 26 edits. However, getting 26 edits is high when the maximal length of the traces is of 37 activities.

To improve the search of multi-alignments, we propose two methods. First one is to randomly select some traces and look for a multi-alignment of these selected traces only. Second we preprocess the event log such that traces that are similar are grouped together. A very simple preprocessing step is the use of Kmean algorithm on the frequencies of the activities per trace. Each log sequence is transformed into a vector where the dimensions represent the activities. Then sequences that have close frequency of the same activities are grouped together by the Kmeans algorithm. Once the groups are created, we discover a multi-alignment per cluster. We give in Appendix. B.2 the script of this experiment.

Results. Tab. 3.5 presents the results in term of maximal distance between the found multi-alignments and the log sequences (the multi-alignments for 3 clusters are given in Appendix. B.2). Because one multi-alignment is learned per group of traces for the random selection and Kmean preprocessing methods, we give the maximum and average of the multi-alignment distances. We observe, from the average column, that randomly selecting some traces can slightly help to get better model-based trace variants. However the gain of quality is low. This is due to the high number of trace variants that disables homogeneous groups of traces.

By using the Kmean method to partition the log sequences, we improve a lot the quality of the multi-alignments. For 3 clusters, thus providing three multi-alignments, the maximal distance between the discovered representatives and the log traces is 10. We can reduce this distance by increasing the number of clusters. Then, for process instances analysis, these artefacts can stand for model-based trace variants. They are associated to a maximal distance which can easily be interpreted for decision makings.

Experiment Conclusion

By partitioning the log sequences with a basic clustering algorithm, we discovered good quality multi-alignments. Then, a multi-alignment is learned per cluster and stands for model-based trace variants. The maximal distances of the multi-alignments to their corresponding log traces can be used as an explicit key indicator for business decisions.

3.6 Conclusion

The drawn idea of multi-alignments is to obtain a modeled sequence that represents a set or a subset of log sequences. Used as model-based trace variants, multi-alignments have the advantages to be included in the existing process model of organizations and to be associated to the maximal distance of its corresponding log sequences. Thanks to the formal definitions and the optimal algorithm presented in this chapter, organizations obtain a reliable and explainable solution for representing several process instances.

We presented two algorithms and several heuristics for computing multi-alignments which allow different uses. Our SAT algorithm specifically encodes the definitions as it is and gives optimal solutions. Due to the complexity of the SAT formulas, we developed an A*-based algorithm to approach optimal results which requires much less memory. By experimenting the two methods and their heuristics on small and large instances, we observed a small quality difference and a runtime improvement. The presented algorithms work for the different conformance checking artefacts which provides a good understanding of their relationships.

Observing the experimentation, we note that a unique multi-alignment badly represents the entire log due to the different behaviors that exist in systems. For this purpose, we recommend the use of a classical clustering method, like Kmean, as a preprocessing step in order to group the log traces by similarity and thus find better quality multi-alignments. Then the clusters hold for subsets and one gets a model-based trace variants per cluster.

However, this technique is limited because the proposed preprocessing methods usually use frequency of the the activities in sequences instead of the sequential information of the process instances. From this last remark, another idea emerges: clustering the log traces directly through the known process model of organizations. By doing so, we ensure to group the traces by behaviors.

Chapter 4

Model-based Clustering of Log Traces through Alignments

Chapter Overview

In the previous chapter, we presented the multi-alignment artefact that allows one to get a unique model-based representative of a log. However, the log traces can differ while being fitting to the process model. A unique model-based variant is then not accurate in this situation.

Clustering methods are unsupervised methods, i.e., there are no predefined classes, that aim at grouping objects by similarities. In this chapter, we present model-based clustering methods of process instances. Thanks to alignments, we can relate log traces to the associated model and extract parts of it that represent the clusters. We propose three kinds of model-based variants: full runs, processes and subnets of the initial process model.

The chapter is organized as follows. First, we present the motivation of the work with an artificial example. Then, in section 4.2, we trace the state-of-the-art methods for clustering process instances but also the works on log variant extraction because we pretend that our methods provides model-based trace variants. The first alignment-based clustering method has been introduced in [33] and is also part of the related work. Section 4.3 gives the definitions of the alignment-based clustering methods. In Section 4.4 we proudly present quality criteria of the clustering methods. Furthermore, some theoretical properties relate the different types of clustering and are given in Section 4.4. Then, we present in Section 4.5 the complexity to obtain the clusterings. Section 4.6 details the implementations. We propose a SAT encoding and a sampling approach to deal with large event logs. The latter is strengthened by statistical confidences. Finally, the chapter contains an experimentation section with qualitative and quantitative experiments. In the subsection 4.8.4, we give a case study and compare our model-based trace variants to another method of the literature. Section 4.9 concludes the chapter.

4.1 Motivation

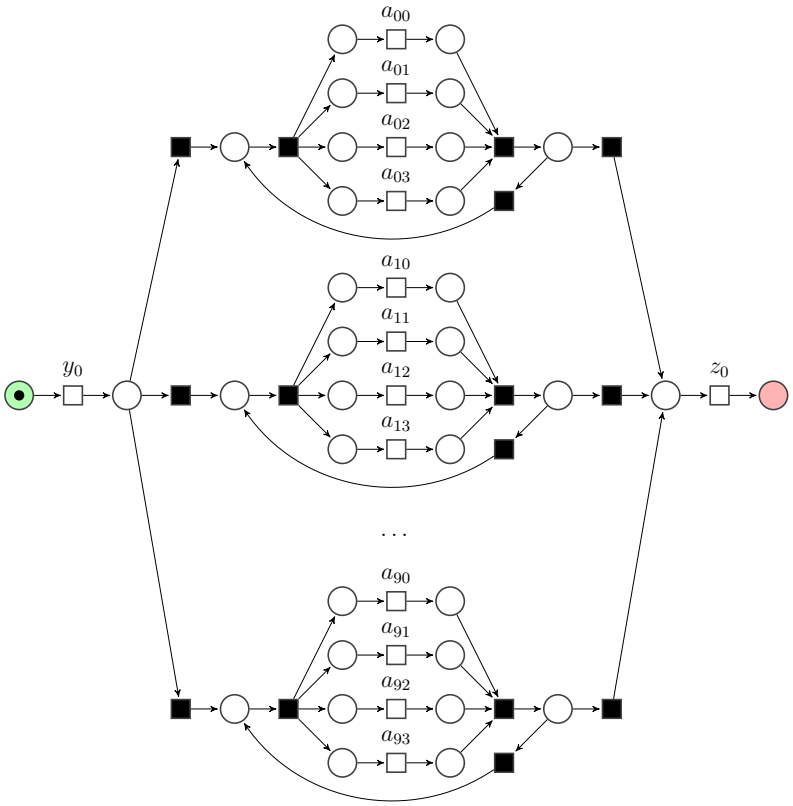


Figure 4.1: Process Model of Motivation Example for Model-based Clustering

Let us illustrate the main message of this section with the process represented by the model shown in Figure 4.1. This is a synthetic process where, after some common activity is executed, 10 branches are possible, each one enabling at isolation concurrent and loop behavior. Once the executed branch terminates, a common activity is executed. Although this process has a very clear structure, it is very likely that an event log for this process models has a high number of trace variants. In fact, after simulating the model to obtain 500 traces, we obtained 411 trace variants in any of the commercial tools available. However, if subnet centroids are used as a generalized notion of variants, 10 variants are considered (each for each subnet a_i, for $0 \leq i \leq 9$). Then, the analysis of the process instances becomes much easier for decision makers. Hence, the motivation of this chapter is to extract parts of the model that contains choice, concurrency and loops behaviors to represent the different groups of behaviors contained in logs.

$$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0 \rangle$$
$$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$$
$$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0 \rangle$$
$$\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0 \rangle$$
$$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$$
$$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$$
$$\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$$
$$\langle y_0, a_{01}, a_{01}, a_{02}, a_{02}, a_{03}, z_0 \rangle$$

Figure 4.2: Artificial Log associated to Model of Fig. 4.1

4.2 Related Work

The related work is divided in three sections. First, we provide the state-of-the-art clustering methods for log sequences. Then, we give the other works that tackle the notion of model-based variants of log traces. Finally, an alignment-based clustering has been started before this thesis project and belongs to the literature.

4.2.1 Log Traces Clustering

In the two last decades, log trace clustering have shown a particular interest in process mining research. A recent review by Zandkarimi et al. counts 103 relevant research works on the subject published between 2004 and 2020 [144]. The article wraps the contributions of the literature in a generic framework given in Fig. 4.3.

The first difference in the works of the literature is the perceptive view of the clustering approach. Some works focus on the *control-flow* of the processes [21, 63, 78, 67, 39, 58, 24] where others address the *context perspective* that refers to the available data contained in events like the performance and timestamps [133]. Some works involve both perspectives [108, 79]. We observe that the distribution of works matches with the building blocks given in the framework where the control-flow perspective, that contains most of the contributions, stands in the Fundamental box.

Control-flow methods handle the sequences of activities. Again, the works can be partitioned in sub-groups. On the first hand, the sequences are given in *vector spaces*. A classical approach is to use bag of words where the activities are the dimensions of the vectors [108]. However, these methods lack the sequential information contained in log. To overcome this limit, works like [63, 39, 21, 78] generate features based on patterns that serve as dimensions of the vector space. The simplest but interesting approach is to use k-grams, i.e., the sub-sequences of k activities as dimensions [63, 39]. In [78], Lu et al. use the Frequent Sequence Patterns over the log that they obtained with the CloFAST algorithm [61]. Likewise, Bose et al. formally define six complex feature sets that allow tackling patterns that are conserved across the traces [21]. The features are based on repetition of sub-sequences in log.

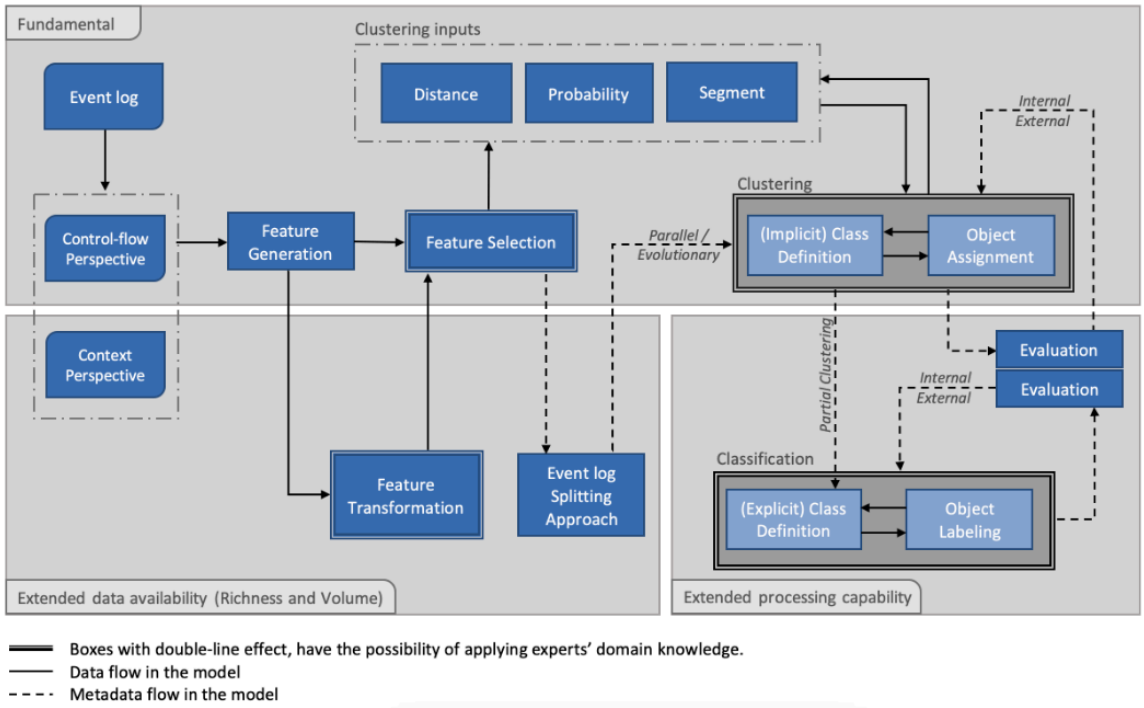


Figure 4.3: Generic Framework for Trace Clustering in Process Mining presented by [144]

Once the vector spaces are set, many data mining techniques are enabled. The most used approach is k-mean [80] but there are also the use of a SVM (Support Vector Machines [89]) approach in [46] and a hierarchical clustering in [133]. In [78], the clustering borders classification with the definition of scores of the traces to belong to a cluster, based on the created features.

In the other hand of control-flow approaches, the log sequences are kept *as-is*. The techniques involve syntactic distance functions between sequences like the Hamming distance or more commonly some edit distances like the Levenshtein distance as it is use in alignment [22, 55].

Finally some works tackle *model-based* structures for log trace clustering. Inspired from bioinformatics, Cadez et al. [24], and later Ferreira et al. [58] and Hompes et al. [67], use Markov chains as a baseline of their approach to partition the log sequences. This corresponds to the Probability blue box of clustering inputs shown in Fig. 4.3. In some works, a preliminary matrix of trace similarity is constructed before the clustering approach [47, 67]. Furthermore, in [138], process models are taken as a baseline in the process of clustering. Some intermediate models are computed to group the log traces.

The motivations for log trace clustering diverge in the literature. While in most works the aim is to discover more conform process models in term of fitness or complexity [21, 63, 138], Hompes et al. study deviation detection in log with clustering [67].

In practice, we observe that clustering approaches are welcomed in healthcare [47, 78] and web services [58, 24]. One main limit of the works that have been highlighted in real-

life situation is the scalability of the methods. Lu et al. indicate in [78] the need to treat hundreds of thousands of patients and millions of events per year.

Perhaps the closest clustering work to our is [39] where the results of the clustering are given with *Super-Instances* that are representatives of the discovered groups of traces. The method creates k-grams and use a PCA (Principal Component Analysis [3]) technique combined with the K-means algorithm, thus using the Euclidean distance. The Super-Instances are then mean of the input log, and can be used as representative, i.e., variants. In the experimental evaluation, we provide a detailed comparison with this closest work.

4.2.2 Model-based Variants of Log Traces

A complete and detailed review of trace variant analysis can be found in [115]. The work by van Beest et al. [116] relies on the product automaton of two event structures to distill all the behavioral differences between two process variants from the respective event logs, and render these differences to end users via textual explanations. Cordes et al. [36] discover two process models and their differences are defined as the minimum number of operations that transform on model to the other. This work was extended in [11] to compare process variants using annotated transition systems.

Pini et al. [90] contribute a visualization technique that compares two discovered process models in terms of performance data. The work in [143] proposes an extension of this work, by considering a normative process model alongside with event logs as inputs, and adding more data preparation facilities. This work is the only one in the literature that like us, considers the process model as input, but it uses the process model merely to compute alignments with the aim of computing performance information.

The works by Bolt et al. and Nguyen et al. are grounded on statistical significance. Bolt et al. [16] use an annotated transition system to highlight the differences between process variants. The highlighted parts only show different dominant behaviors that are statistically significant with respect to edge frequencies. This work was later extended in [17], by inducting decision trees for performance data among process variants. Nguyen et al. [88] encode process variants into *Perspective Graphs*. The comparison of perspective graphs results in a *Differential Graph*, which is a graph that contains common nodes and edges, and also nodes and edges that appear in one perspective graph only.

The use of an explicit characterization of concurrency has been considered recently in process discovery: the works in [93, 92] show how to improve the discovery of a process model by folding the initial unfolding that satisfies the independence relations given as inputs. In the area of conformance checking, the same phenomena has been observed: the work in [77] assumes traces are represented as partial order, thus allowing again an explicit characterization of concurrency in the problem formalization.

Perhaps the works more similar to the one of this chapter are [45, 83], where a transition system representing the event log is clustered, so that a set of simpler process models is generated. Tailored state-based properties that guarantee certain Petri net classes are used to guide the clustering, whereas in this work the computation of subnets is unrestricted.

Full Runs Centroids	Traces
$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0\rangle$	$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0\rangle$ $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$
$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0\rangle$	$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0\rangle$
$\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$	$\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$
$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$	$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$
$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$	$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$ $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$
nc	$\langle y_0, a_{01}, a_{01}, a_{02}, a_{02}, a_{03}, z_0\rangle$

Table 4.1: Example of Alignment-based Trace Clustering (ATC) of Log Traces Contained in Log of Fig. 4.2 for a Maximum Allowed Distance to 2.

Last but not least, Tax et al. finds in [110] several local process models that depict patterns contained in log.

All the aforementioned algorithms consider only the event log as input, in this thesis, we use an existing process model as a baseline of our variant extraction.

4.2.3 ATC: Alignment-based Clustering

This chapter is the continuity of alignment-based trace clustering introduced in 2017 in [33]. Alignment-based trace clustering is a particular form of trace clustering that relies on a model N of the observed system. The model may very well be incomplete or imperfect but it will however give a guideline for the clustering of the observed traces. The idea of alignment-based trace clustering is to explicit the relation between log traces and full runs of N. Concretely, each cluster of log traces will be assigned a full run u of N, presented as the centroid of the cluster. Hence, traces in the same cluster are not only similar among them, but they are related to a run of the model, which together validates a part of the model and explains the observed log traces.

Definition 17 (Alignment-based Trace Clustering (ATC))**.** *For a log L and a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$, an alignment-based trace clustering of L w.r.t. N is a tuple $\mathcal{C} = \langle\{u_1 \dots u_m\}, \chi\rangle$ where $u_1 \dots u_m$ ($m \in \mathbb{N}$) are full runs of N which serve as centroids for the clusters and $\chi : L \to \{\mathtt{nc}, u_1 \dots u_m\}$ maps log traces either to the centroid of its cluster $\chi(\sigma)$, or to none of the clusters, denoted by* nc*.*

Each set $\chi^{-1}(u_i)$, for $i \in \{1 \dots m\}$, defines the cluster whose centroid is u_i. The set $\chi^{-1}(\mathtt{nc})$ contains the traces which are left non-clustered.

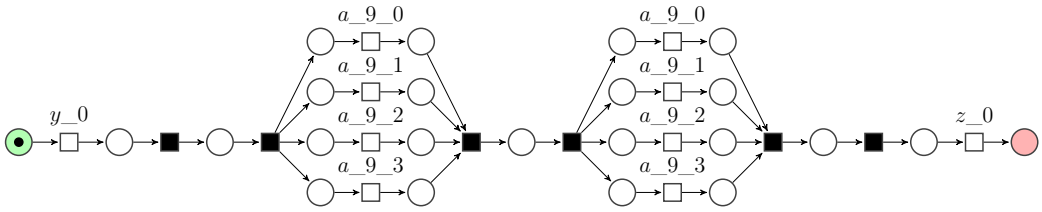


Figure 4.4: Example of a Process of the Process Model in Fig. 4.1

Example 4.2.1 (Alignment-based Clustering)**.** *Tab. 4.1 shows a clustering of the traces of the log of Fig. 4.2 based on the model of Fig. 4.1. The maximal Levenshtein distance between the full runs centroids and the traces of the corresponding cluster is to 2. The centroids can be hold as good variants for representing the log traces, thus reducing the set of variants to focus on.*
The last log trace is associated to `nc` *because it cannot be aligned to a run of N for a distance lower or equal to 2.*

4.3 Fitting Centroids to Concurrency and Loop Behavior

As presented above, full runs of process models can be extracted to represent log traces by using Alignment-based Trace Clustering (ATC). In this section, we want to go further and allow one to extract model-based trace variants aware of concurrent and loop behaviors contained in the process model.

4.3.1 APOTC : Alignment and Partial Order based Trace Clustering

In full runs of a process model, transition occurrences are totally ordered. However transitions can be handled in different orders for the same process in case of concurrency. In the model of Fig.4.2 traces $\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$ and $\langle y_0, a_{12}, a_{11}, a_{13}, a_{10}, z_0\rangle$ follow the same process but differ by the order of the transitions.

They can however be seen as two linearizations of a common representation based on partial-order runs which represents a *process*.

Definition 18 (Partial-Order Representation of Runs: Process [52])**.** *A (non-branching) process $\mathcal{P}$ of a Petri Net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$ is a tuple $\mathcal{P} = \langle B, E, G, B_0, B_f, h\rangle$ where:*

- *(B, E, G, B_0, B_f) is a non-branching, finite, acyclic Petri Net, i.e.*

– *its causality relation* G^+ *is acyclic, and*

– *it has no forward and no backward branchings:*

$$\begin{array}{lll} \forall b \in B & \exists! e \in E \cup \{\bot\} & b \in e^\bullet \\ \forall b \in B & \exists! e \in E \cup \{\top\} & b \in {}^\bullet e \end{array}$$

where $\bot$ *and* $\top$ *are virtual events satisfying* $\bot^\bullet \stackrel{\text{def}}{=} B_0$ *and* ${}^\bullet\top \stackrel{\text{def}}{=} B_f$.

- $h : (B \cup E) \to (P \cup T)$ *is a function that maps the non-branching process* $\mathcal{P}$ *in the Petri Net* N *with the following relations:*

  – $h(B) \subseteq P$ *and* $h(E) \subseteq T$

  – $\forall e \in E, h_{|{}^\bullet e}$ *is a bijection between* ${}^\bullet e$ *and* ${}^\bullet h(e)$*, same reasoning for* $h_{|e^\bullet}$

  – $h_{|B_0}$ *is a bijection between* B_0 *and* m_0*, likewise for* $h_{|B_f}$

We write $Runs(\mathcal{P})$ for the set of full runs of the process $\mathcal{P}$. For every full run $\langle e_1 \dots e_n \rangle$ of a process $\mathcal{P}$ of a Petri net N, the sequence $u \stackrel{\text{def}}{=} \langle h(e_1) \dots h(e_n) \rangle \in T^*$ is called a *linearization* of $\mathcal{P}$. Every linearization of $\mathcal{P}$ is a full run of N.

Example 4.3.1 (Processes)**.** *Fig. 4.4 shows a process of the Petri net in Fig. 4.1. This process represents both full runs* $\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$ *and* $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$ *which differ only by the order of concurrent transitions. Similarly,* $\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0 \rangle$ *and* $\langle y_0, a_{12}, a_{11}, a_{13}, a_{10}, z_0 \rangle$ *are also represented by a single process.*

By using processes as model-based trace variants, traces that only differ by the order of concurrent activities can be grouped and represented by the same variant. The process presented in Fig. 4.4 is then a unique variant for the log traces $\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$ and $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$. Hence, processes appear as a good representation for trace variants aware of concurrency.

Similarly to ATC, process representatives can be found with a clustering method which gives process centroids that serve as model-based trace variants.

Definition 19 (Alignment and Partial Order based Trace Clustering (APOTC))**.** *An alignment and partial order based trace clustering (APOTC) of a log* L *w.r.t. a Petri net* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$*, is a tuple* $\mathcal{C} = \langle \{\mathcal{P}_1 \dots \mathcal{P}_m\}, \chi \rangle$ *where* $\mathcal{P}_1 \dots \mathcal{P}_m$ *(*$m \in \mathbb{N}$*) are processes of* N *which serve as centroids for the clusters and* $\chi : L \to \{\mathtt{nc}, \mathcal{P}_1 \dots \mathcal{P}_m\}$ *maps log traces either to the centroid of its cluster* $\chi(\sigma)$*, or to none of the clusters, denoted by* `nc`*.*

Like in the ATC method, the distance between traces and model-based trace variants, is minimized. The distance between a trace σ and a process $\mathcal{P}$ is defined by $dist_{\mathcal{P}}(\sigma, \mathcal{P}) \stackrel{\text{def}}{=} \min_{u \in Runs(\mathcal{P})} dist(\sigma, u)$, where *dist* is usually the Levenshtein edit distance $\mathcal{L}$. Then distances of APOTC minimize the following.

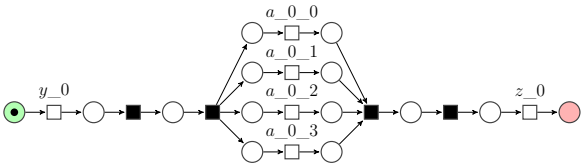
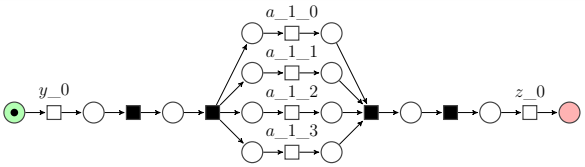
Processes as Model-based Trace Variants	Traces
	$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0\rangle$ $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$
	$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0\rangle$ $\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$
process of Figure 4.4	$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$ $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$
process similar to Figure 4.4, not represented	$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$
nc	$\langle y_0, a_{01}, a_{01}, a_{02}, a_{02}, a_{03}, z_0\rangle$

Table 4.2: Example of APOTC of Traces Contained in the Log of Fig. 4.2. Maximum Allowed Distance Between Clustered Traces and their Centroids is 2.

$$\max_{\sigma \in L, \chi(\sigma) \neq \mathtt{nc}} dist_{\mathcal{P}}(\sigma, \chi(\sigma)) \tag{4.1}$$

Example 4.3.2 (APOTC)**.** *Tab. 4.2 shows an APOTC of log traces of Fig. 4.2 based on the model of Fig. 4.1.*

By using processes as model-based trace variants, one merges clusters which were separated in the ATC of Tab. 4.1 and identify richer variants.

4.3.2 AMSTC: Alignment and Model Subnet-based Trace Clustering

APOTC separates process arising from traces corresponding to different number of loop iterations, e.g., the traces $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$ and $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$. The issue is due of the fixed size of runs of processes, which do not represent loops. To overcome this limitation, we introduce subnets of models.

Definition 20 (Subnet of Petri net)**.** *A* subnet *of a Petri net* $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda\rangle$ *is a Petri net* $\langle P, T', F_{|T'}, m_0, m_f, \Sigma_{|T'}, \lambda\rangle$ *with* $T' \subseteq T$*, and* $F_{T'} \stackrel{\text{def}}{=} F \cap (P \times T' \cup T' \times P)$*.*

Example 4.3.3 (Subnet)**.** *Fig. 4.5 presents a subnet of the model of Fig. 4.1.*

Observe that our definition of subnets, based on selecting transitions, restricts the semantics of the net and cannot produce new behaviors. Formally:

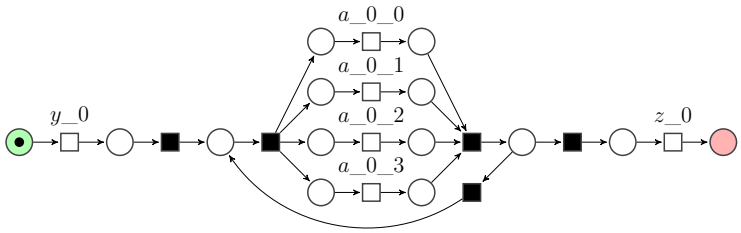


Figure 4.5: A Subnet of the Process Model in Fig. 4.1.

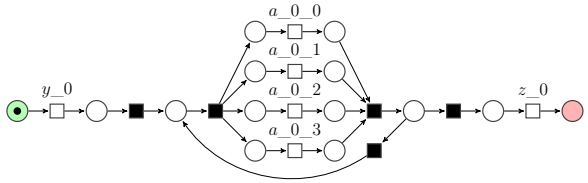
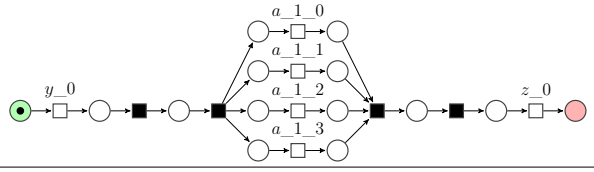
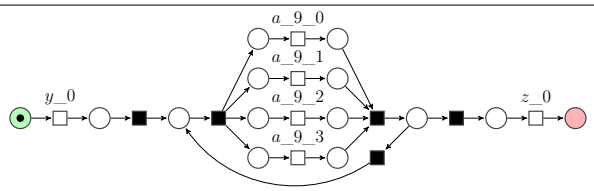
Subnets as Model-based Trace Variants	Traces
	$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0 \rangle$ $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$ $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$
	$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0 \rangle$ $\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0 \rangle$
	$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$ $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$
nc	$\langle y_0, a_{01}, a_{01}, a_{02}, a_{02}, a_{03}, z_0 \rangle$

Table 4.3: Example of Alignment and Model Subnet-based Trace Clustering (AMSTC) of Traces Contained in Log of Fig. 4.2 for a Maximum Allowed Distance to 2.

Lemma 6. *Every full run (resp. process) of a subnet of a Petri net N, is a full run (resp. process) of N.*

Definition 21 (Alignment and Model Subnet-based Trace Clustering (AMSTC))**.** *For a log L and a Petri net $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, an alignment and model subnet trace clustering, of L w.r.t. N is a tuple $\mathcal{C} = \langle \{\mathcal{N}_1 \dots \mathcal{N}_m\}, \chi \rangle$ where $\mathcal{N}_1 \dots \mathcal{N}_m$ are subnets of N which serve as centroids for the clusters and $\chi : L \to \{\texttt{nc}, \mathcal{N}_1 \dots \mathcal{N}_m\}$ maps log traces either to the centroid of its cluster $\chi(\sigma)$, or to none of the clusters, denoted by* nc*.*

Distances between a clustered trace σ and its centroid $\mathcal{N}$, used as model-based trace variants, is defined as $dist_{\mathcal{N}}(\sigma, \mathcal{N}) \stackrel{\text{def}}{=} \min_{u \in Runs(\mathcal{N})} dist(\sigma, u)$, with *dist* a distance between sequences, usually the Levenshtein edit distance in process mining. Computing this distance corresponds to align traces to model. Alignment criterion of an AMSTC minimizes equation (4.2).

$$\max_{\sigma \in L, \chi(\sigma) \neq \mathtt{nc}} dist_{\mathcal{N}}(\sigma, \chi(\sigma)) \tag{4.2}$$

Example 4.3.4 (AMSTC)**.** *Tab. 4.3 shows an AMSTC of log traces of Fig. 4.2 based on the model of Fig. 4.1. Traces* $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$ *and* $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0 \rangle$ *are now clustered together.*

The novel variant forms allow traces that only differ on concurrency and loops according to the model to be grouped together. Business analysis is sometimes too complicated due to the number of variants. With this approach, we help one to reduce the number of variants the stakeholder has to understand.

4.4 Quality Criteria of Clusterings

In this section, we provide criteria which contribute in the qualification of a good clustering. Furthermore, we provide properties that relate presented clustering methods.

4.4.1 General Criteria Definition

First, we present the following list of criteria that we have identified to determine the quality of clustering:

- $\boldsymbol{d(\mathcal{C})}$*, maximum distance between a trace and the centroid of its cluster*: this criterion, defined by $\max_{\sigma \in L \setminus \chi^{-1}(\mathtt{nc})} dist(\sigma, \chi(\sigma))$, where *dist* is a distance function that depends on the type of clustering, will be minimized to increase the fit of the centroids to their traces. In case of a log containing noise, a small distance may induce many non-clustered traces.

- $\boldsymbol{\Delta(\mathcal{C})}$*, sum of distances* : the sum $\Delta(\mathcal{C}) \stackrel{\text{def}}{=} \sum_{\sigma \in L \setminus \chi^{-1}(\mathtt{nc})} dist(\sigma, \chi(\sigma))$, with *dist* depends on the type of clustering, can be seen as a variant or a refinement of the previous criterion $d(\mathcal{C})$. It will also be minimized in order to get the most representative centroids.

- $\boldsymbol{n(\mathcal{C})}$*, number of clusters* : The number of clusters provides an interesting perspective, which is analogous to the number of trace variants of a process model, but in this case from the log perspective.

- $\boldsymbol{\check{c}(\mathcal{C})}$*, ratio of clustered traces:* this ratio, defined as $\check{c}(\mathcal{C}) \stackrel{\text{def}}{=} \frac{|L| - |\chi^{-1}(\mathtt{nc})|}{|L|}$, is close to 1 for a process model that covers most of the behavior of the log. $\check{c}(\mathcal{C})$ also highlights the ratio of distant traces, i.e. traces that deviate from the model, for a given maximum distance $d(\mathcal{C})$.

- $\boldsymbol{\Phi(\mathcal{C})}$, *inter-cluster distance* : the distance between the centroids is also an important parameter. A larger distance involves distant clusters, this is why this parameter should be maximized in order to prevent overlay between the clusters.

For an ATC, $\mathcal{C} = \langle\{u_1 \dots u_n\}, \chi\rangle$, the inter-cluster distance is defined between run centroids as $\Phi(\mathcal{C}) \stackrel{\text{def}}{=} \min_{i \neq j} dist(u_i, u_j)$. The criterion slightly varies for APOTC and AMSTC where centroids are respectively processes and subnets. For an APOTC, $\mathcal{C} = \langle\{\mathcal{P}_1 \dots \mathcal{P}_n\}, \chi\rangle$ we obtain: $\Phi(\mathcal{C}) = \min_{i \neq j} dist(\mathcal{P}_i, \mathcal{P}_j)$, where the appropriate notion of distance between processes is: $dist(\mathcal{P}, \mathcal{P}') \stackrel{\text{def}}{=} \min_{\substack{u \in Runs(\mathcal{P}) \\ u' \in Runs(\mathcal{P}')}} dist(u, u')$. Similarly, for AMSTC, $\mathcal{C} = \langle\{\mathcal{N}_1 \dots \mathcal{N}_n\}, \chi\rangle$, we have $\Phi(\mathcal{C}) = \min_{i \neq j} dist(\mathcal{N}_i, \mathcal{N}_j)$ where $dist(\mathcal{N}, \mathcal{N}') \stackrel{\text{def}}{=} \min_{\substack{u \in Runs(\mathcal{N}) \\ u' \in Runs(\mathcal{N}')}} dist(u, u')$.

The detailed criteria are inspired from the Data Mining domain [62, 13]. Other measures like the Dunn [50], which compares distances between items that share or not a cluster, and the Silhouette [103], that computes if items is close enough to their clusters instead of the others, help the user to analyze its clustering. As usual when multiple parameters are taken into account, there will not exist in general a unique clustering optimizing all the criteria together. Instead, every clustering problem should consider a good balance between the parameters to optimize.

Example 4.4.1 (Quality of clustering)**.** *We present the quality results of the three clustering presented in Tables 4.1, 4.2 and 4.3:*

Method	$\boldsymbol{d(\mathcal{C})}$	$\boldsymbol{\Delta(\mathcal{C})}$	$\boldsymbol{n(\mathcal{C})}$	$\boldsymbol{\check{c}(\mathcal{C})}$	$\boldsymbol{\Phi(\mathcal{C})}$
ATC	*2*	*4*	*5*	*7/8*	*4*
APOTC	*0*	*0*	*4*	*7/8*	*4*
AMSTC	*0*	*0*	*3*	*7/8*	*8*

We observe that more general centroids help in fitting the traces with the criterion $\boldsymbol{d(\mathcal{C})}$ *that drops from 2 to 0 when using processes or subnets as centroids.* $\boldsymbol{\Delta(\mathcal{C})}$ *is highly correlated to* $\boldsymbol{d(\mathcal{C})}$ *which is well drawn in this example.*
The criteria $\boldsymbol{n(\mathcal{C})}$ *and* $\boldsymbol{\Phi(\mathcal{C})}$ *are of great interest in this example. First, we observe that we reduce the number of clusters which implies less variants for analysis. Also, the clusters present more differences when using subnets as centroid. Indeed all the concurrent and loop behaviors are grouped in the same cluster. This is highlighted by the high value of the inter-cluster distance* $\boldsymbol{\Phi(\mathcal{C})}$ *of the AMSTC method.*

Moreover, even between the same type of clustering, we can obtain different results depending on the criteria that are optimized.

Example 4.4.2 (Another example of ATC, not optimizing the same criteria)**.** *The next table shows another examples of ATC, where no trace is left unclustered, thus optimizing* $\check{\mathbf{c}}(\boldsymbol{\mathcal{C}})$*. However, the quality criteria* $\boldsymbol{\Delta}(\boldsymbol{\mathcal{C}})$ *and* $\boldsymbol{d}(\boldsymbol{\mathcal{C}})$ *are not as optimized as in the clustering presented in Tab 4.1. Here* $\boldsymbol{d}(\boldsymbol{\mathcal{C}}) = 3$*.*

Full Runs Centroids	*Traces*
$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0\rangle$	$\langle y_0, a_{00}, a_{01}, a_{02}, a_{03}, z_0\rangle$ $\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$ $\langle y_0, a_{01}, a_{01}, a_{02}, a_{02}, a_{03}, z_0\rangle$
$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0\rangle$	$\langle y_0, a_{12}, a_{10}, a_{13}, a_{11}, z_0\rangle$
$\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$	$\langle y_0, a_{13}, a_{11}, a_{12}, a_{10}, z_0\rangle$
$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$	$\langle y_0, a_{01}, a_{00}, a_{02}, a_{03}, a_{01}, a_{00}, a_{02}, a_{03}, z_0\rangle$
$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$	$\langle y_0, a_{90}, a_{91}, a_{93}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$ $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0\rangle$

Intra-Cluster Distance for AMSTC

If applied unrestricted, AMSTC can use as centroids, subnets with branchings and loops, and then cluster together very different log traces. The intra-cluster distance aims at controlling this aspect. For instance, taking as centroid the complete net of Fig. 4.1 would not yield a satisfactory AMSTC. Instead, traces in the same cluster should be similar and represent a generalized notion of trace variant. This criterion is quantified by the *intra-cluster distance* $\Theta(\mathcal{C})$. Clusterings with low $\Theta(\mathcal{C})$ will be preferred.

- $\boldsymbol{\Theta(\mathcal{C})}$, *the intra-cluster distance*: Before defining the intra-cluster distance of a clustering $\mathcal{C}$, we focus on each of its centroids separately: for every centroid $\mathcal{N}_k$, define

$$\Theta'(\mathcal{N}_k) \stackrel{\text{def}}{=} \sup_{\mathcal{P},\mathcal{P}' \in Proc(\mathcal{N}_k)} \frac{dist(\mathcal{P},\mathcal{P}')}{(1+\epsilon)^{max(|\mathcal{P}|,|\mathcal{P}'|)}} \tag{4.3}$$

  where $|\mathcal{P}|$ denotes the number of events in $\mathcal{P}$, and $\epsilon > 0$ is a parameter set by the user in order to limit (more or less) the influence of long processes. Indeed, when the subnet $\mathcal{N}_k$ has loops, it has infinitely many processes, arbitrary large, which yields arbitrary large distance to the smaller processes. Yet, such subnets may be relevant, as illustrated by Example 4.5. This is why our definition penalizes more for distances between small processes.

  Finally, the intra-cluster distance of a clustering $\mathcal{C} = \langle\{\mathcal{N}_1 \ldots \mathcal{N}_n\}, \chi\rangle$ is:

$$\Theta(\mathcal{C}) \stackrel{\text{def}}{=} \max_k \Theta'(\mathcal{N}_k) \tag{4.4}$$

Example 4.4.3 (Epsilon Interest in the Intra-cluster Distance)**.** *Let consider the subnet N_s of Fig 4.5. By using the loop we can find novel processes, thus raising the distance between the processes of the subnet $dist(\mathcal{P}, \mathcal{P}')$. However, the length of the processes penalizes the intra-cluster distance. For ϵ high like $\epsilon = 0.2$, we observe that having a maximal length of 17 transitions (by unfolding once the loop like in Fig 4.4), we can obtain a maximum $dist(\mathcal{P}, \mathcal{P}') = \frac{8}{1.2^{17}} = 0.36$ where $\mathcal{P}, \mathcal{P}' \in Proc(\mathcal{N}_f)$. Then, for smaller maximal length, we do not unfold the loop and we get $dist(\mathcal{P}, \mathcal{P}') = \frac{4}{1.2^{10}} = 0.36$.*

4.4.2 Relating APOTC to ATC

Full runs of Petri nets have their process representations:

Definition 22 (Process representation of a full run)**.** *Every full run u of a (safe) model N induces a process of N. This process is unique up to isomorphism [52] and is denoted by $\Pi(u)$.*

Example 4.4.4 (Π isomorphism)**.** *Fig. 4.4 shows the process representation of $\langle y_0, a_{90}, a_{93}, a_{91}, a_{92}, a_{90}, a_{91}, a_{93}, a_{92}, z_0 \rangle$.*

Then any ATC can be casted as an APOTC. All the full runs centroids of an ATC, which are sequential executions, can be represented as processes using Def. 22. The following theorem explains how this transformation affects the quality criteria of the clusterings.

Theorem 1. *For any ATC $\mathcal{C}_u = \langle \{u_1 \dots u_n\}, \chi_u \rangle$, we define $\forall i \in \{1 \dots n\}$ $\mathcal{P}_i \stackrel{\text{def}}{=} \Pi(u_i)$ and $\chi_{\mathcal{P}} \stackrel{\text{def}}{=} \Pi \circ \chi_u$ (by convention $\Pi(\texttt{nc}) = \texttt{nc}$) inducing $\mathcal{C}_{\mathcal{P}} = \langle \{\mathcal{P}_1 \dots \mathcal{P}_n\}, \chi_{\mathcal{P}} \rangle$ its corresponding APOTC of the same process model N and the same log L. The distances below follow the properties:*

1. *$d(\mathcal{C}_u) \geq d(\mathcal{C}_{\mathcal{P}})$ and $\Delta(\mathcal{C}_u) \geq \Delta(\mathcal{C}_{\mathcal{P}})$ with equality if the model is sequential*
2. *$\Phi(\mathcal{C}_u) \geq \Phi(\mathcal{C}_{\mathcal{P}})$ with equality if the model is sequential*
3. *$n(\mathcal{C}_u) = n(\mathcal{C}_{\mathcal{P}})$ and $\check{c}(\mathcal{C}_u) = \check{c}(\mathcal{C}_{\mathcal{P}})$*

Proof. We first observe that the obtained set $\{\mathcal{P}_1 \dots \mathcal{P}_n\}$ is by Def. 22 a set of subnets of N and $\chi_{\mathcal{P}}$ maps every clustered log traces to a subnet and non-clustered log traces to $\texttt{nc}$. Then $\mathcal{C}_{\mathcal{P}} = \langle \{\mathcal{P}_1 \dots \mathcal{P}_n\}, \chi_{\mathcal{P}} \rangle$ is indeed an APOTC.

1. Every trace σ of L is either clustered ($\chi_u(\sigma) = u_i,\ i \in \{1 \dots n\}$) or non-clustered ($\chi_u(\sigma) = \texttt{nc}$). The maximum distance between traces and centroids $d(\mathcal{C}_u)$ depends only on clustered traces: $\forall \sigma \in L_{\setminus \chi_u{}^{-1}(nc)} \quad dist(\sigma, \chi_u(\sigma)) \leq d(\mathcal{C}_u)$. By Def. 22 $\chi_u(\sigma) \in Runs(\chi_{\mathcal{P}}(\sigma))$. Then for any clustered trace σ, we have $d(\mathcal{C}_{\mathcal{P}}) \leq dist(\sigma, \chi_{\mathcal{P}}(\sigma)) \leq dist(\sigma, \chi_u(\sigma)) \leq d(\mathcal{C}_u)$ with equality if the model is sequential (no other run in $Runs(\chi_{\mathcal{P}}(\sigma))$). Furthermore, $\Delta(\mathcal{C})$ is the sum of the distances: $\Delta(\mathcal{C}_{\mathcal{P}}) \leq \Delta(\mathcal{C}_u)$.

2. Let u_i and u_j, $i, j \in \{1 \dots n\}$, be two centroids of the ATC. The corresponding processes of those centroids are defined by $\mathcal{P}_i = \Pi(u_i)$ and $\mathcal{P}_j = \Pi(u_j)$ and $u_i \in \mathit{Runs}(\mathcal{P}_i)$ and $u_j \in \mathit{Runs}(\mathcal{P}_j)$. This implies $\mathit{dist}(\mathcal{P}_i, \mathcal{P}_j) \leq \mathit{dist}(u_i, u_j)$ with equality if the model is sequential (no other run in the processes). Consequently $\Phi(\mathcal{C}_\mathcal{P}) \leq \min_{i \neq j} \mathit{dist}(u_i, u_j) = \Phi(\mathcal{C}_u)$ with equality if the model is sequential.

3. This is immediate by definition of $\chi_\mathcal{P}$.

□

As a summary, casting an ATC to an APOTC improves the distances between traces and centroids; in contrast, the resulting APOTC may get a lower (i.e. poorer) inter-cluster distance than the ATC. The number of clusters and ratio of clustered traces are preserved.

This means that clusters that were distant in the ATC may become closer in the APOTC, which appears negative when seen from the perspective of good clusterings presenting distant clusters. But, in the other hand, clusters that become closer will typically be those that one precisely wanted to merge because they represent different interleavings of processes.

This is exactly what happens in the running example. Merging clusters then results in a lower number of clusters $n(\mathcal{C})$, which also helps to get a human understandable clustering and facilitates the analysis of the results by decision makers.

4.4.3 Relating AMSTC to APOTC

Similarly to the previous section, we relate processes to subnets.

Definition 23 (Subnet induced by a process)**.** *Every process $\mathcal{P} = (B, E, G, B_0, B_f, h)$ of a model $N = \langle P, T, F, m_0, m_f, \Sigma, \lambda \rangle$, induces a subnet $\mathcal{N}$ of N defined by $\mathcal{N} = \Psi(\mathcal{P}) \stackrel{\text{def}}{=} (P, h(E), G_{|h(E)}, h(B_0), h(B_f))$.*

Then, every APOTC $\mathcal{C}_\mathcal{P}$ induces an AMSTC whose subnet centroids are subnets are defined according to the process centroids of $\mathcal{C}_\mathcal{P}$.

The following theorem relates APOTC and the induced AMSTC, analogously to Theorem 1 for ATC and APOTC.

Theorem 2. *For any APOTC $\mathcal{C}_\mathcal{P} = \langle \{\mathcal{P}_1 \dots \mathcal{P}_n\}, \chi_{\mathcal{P}_i} \rangle$, we define $\forall i \in \{1 \dots n\}\ \mathcal{N}_i \stackrel{\text{def}}{=} \Psi(\mathcal{P}_i)$ and $\chi_{\mathcal{N}_\mathcal{P}} \stackrel{\text{def}}{=} \Psi \circ \chi_\mathcal{P}$ (by convention $\Psi(\mathtt{nc}) = \mathtt{nc}$) inducing $\mathcal{C}_{\mathcal{N}_\mathcal{P}} = \langle \{\mathcal{N}_1 \dots \mathcal{N}_n\}, \chi_{\mathcal{N}_\mathcal{P}} \rangle$ its corresponding AMSTC of the same process model N and the same log L. The distances below follow the properties:*

1. $d(\mathcal{C}_\mathcal{P}) \geq d(\mathcal{C}_{\mathcal{N}_\mathcal{P}})$ *and* $\Delta(\mathcal{C}_\mathcal{P}) \geq \Delta(\mathcal{C}_{\mathcal{N}_\mathcal{P}})$ *with equality if the model is acyclic*

2. $\Phi(\mathcal{C}_\mathcal{P}) \geq \Phi(\mathcal{C}_{\mathcal{N}_\mathcal{P}})$ *with equality if the model is acyclic.*

3. $n(\mathcal{C}_\mathcal{P}) = n(\mathcal{C}_{\mathcal{N}_\mathcal{P}})$ *and* $\check{c}(\mathcal{C}_\mathcal{P}) = \check{c}(\mathcal{C}_{\mathcal{N}_\mathcal{P}})$

Proof. The correspondence APOTC-AMSTC is similar to the correspondence ATC-APOTC demonstrated in Theorem 1. When properties exactly coincide between ATC and APOTC for sequential models, same results are found between APOTC and AMSTC for acyclic models: without loops, every subnet has a single process and the distances are preserved. □

4.4.4 When AMSTCs meet APOTCs.

Observe that, in our definition of AMSTC, only the behavior of the subnets is considered. Hence, nothing penalizes a clustering for having dead transitions in a cluster, i.e., transitions which do not participate in any full run of the subnet. Intuitively, this situation is not satisfactory since we expect the subnets to give information about the part of the net which really participates in the observed traces. By the way, notice that the subnets induced by processes following Def. 23 never have any dead transition. These subnets also have another property: they all have at least one full run. Let us call *fair* an AMSTC in which every centroid has these two properties. The following theorem establishes a relation between APOTCs and fair AMSTCs.

Theorem 3. *For a log L and an acyclic and trace-deterministic*[1] *model N, the transformation defined in Theorem 2 establishes a bijection from the set of APOTC to the set of fair AMSTCs $\mathcal{C}$ with intra-cluster distance $\Theta(\mathcal{C}) = 0$.*

Proof. Since N is acyclic, for every process $\mathcal{P}$ of N, the subnet induced by $\mathcal{P}$ has no other process than $\mathcal{P}$ itself. This proves that any AMSTC $\mathcal{C}$ obtained from an APOTC has intra-cluster distance $\Theta(\mathcal{C}) = 0$. It is also fair as we noticed earlier.

Now, every centroid $\mathcal{N}_i$ of a fair AMSTC $\mathcal{C}$ with $\Theta(\mathcal{C}) = 0$ has a single process (call it $\mathcal{P}_i$): indeed, since the model is trace-deterministic, every subnet centroid in $\mathcal{C}$ having two different processes would lead to $\Theta(\mathcal{C}) > 0$. This establishes a bijection between the centroids of fair AMSCs with intra-cluster distance 0, and the processes of N, which serve as centroids in APOTCs. This bijection between centroids induces naturally our bijection between APOTCs and AMSTCs.

□

In summary, AMSTC handles both concurrency and repetitive behavior, and under some situations behaves similarly to APOTC.

4.5 Complexity of Alignment-based Trace Clusterings

For a log L and a model N, one is typically interested in computing a trace clustering (ATC, APOTC or AMSTC) $\mathcal{C}$ of L w.r.t. N of sufficient quality, i.e. satisfying some constraints on the quality criteria $d(\mathcal{C})$, $\Delta(\mathcal{C})$, $n(\mathcal{C})$, $\check{c}(\mathcal{C})$. We will see that, at least from a theoretical point

[1] N is *trace-deterministic* if the mapping $u \in \mathit{Runs}(N) \mapsto \lambda(u) \in \Sigma^*$ is injective.

of view, the complexity lies already in the existence of a clustering, and the specification of many quality constraints does not change the complexity.

For a non-empty log L and a model N, there exists an ATC $\mathcal{C}$ of L w.r.t. N having $\check{c}(\mathcal{C}) > 0$ (i.e. such that at least one trace is clustered), iff N has a full run. Indeed, when no constraint is given about the quality criteria $d(\mathcal{C})$, $\Delta(\mathcal{C})$, $n(\mathcal{C})$, $\Phi(\mathcal{C})$…, any full run of N can serve as centroid, and any log trace can be assigned to any cluster. The same holds for APOTC, where centroids are processes of N, since N has a process iff N has a full run; it holds again for AMSTC, taking into account the constraint that any subnet used as centroid should have a full run, or the stronger constraint that the subnet should not have any dead transition, as discussed in Section 4.4.4.

Now, deciding if a model has a full run u, corresponds to checking reachability of the final marking which is in PSPACE-complete or even NP-complete in some case (see Section. 2.3.1).

In practice, relevant clusterings will not use very long full runs (or processes for APOTC) as centroids. Also for AMSTC, no very long full run will be considered in the computation of $d(\mathcal{C})$, $\Delta(\mathcal{C})$ or $\Phi(\mathcal{C})$. Typically, a bound l on the length of the full runs can be assumed, for instance 2 times the length of the longer log trace. Let us call l-bounded a trace clustering satisfying this constraint.

Theorem 4. *The problem of deciding, for a log L, a model N, an integer bound l, integers $d_{\max}$, $\Delta_{\max}$, $n_{\max}$ and a rational number $\check{c}_{\min}$, the existence of a l-bounded ATC (respectively APOTC, AMSTC) $\mathcal{C}$ of L w.r.t. N, having $d(\mathcal{C}) \leq d_{\max}$, $\Delta(\mathcal{C}) \leq \Delta_{\max}$, $n(\mathcal{C}) \leq n_{\max}$ and $\check{c}(\mathcal{C}) \geq \check{c}_{\min}$, is NP-complete.*

Proof. As observed earlier, the problem is NP-hard even with the only constraint that at least one trace is clustered (i.e. $\check{c}(\mathcal{C}) > 0$, or equivalently $\check{c}(\mathcal{C}) \geq \frac{1}{|L|}$). It remains to show that it is in NP: indeed, if there exists a (l-bounded) clustering, there exists one with no more that $|L|$ clusters (forgetting empty clusters cannot weaken the quality criteria); and, by assumption, the size of centroids (defined as $|\sigma|$ for ATC, $|\mathcal{P}|$ for APOTC, number of transitions in the subnet for AMSTC) is bounded by l. So, it is possible to guess a clustering $\mathcal{C}$ in polynomial time. For APOTC and AMSTC, one can also guess in P time the full run $u \in \mathit{Runs}(\chi(\sigma))$, for every clustered trace σ, which will achieve the $\mathit{dist}(\sigma, \chi(\sigma))$. Now, checking that $\mathcal{C}$ satisfies the constraints, only requires to compute Levenshtein's edit distances and minima over sets of polynomial size. This can be done in P time.

□

For ATC, the problem remains in NP with an additional constraint on the inter-cluster distance ($\Phi(\mathcal{C}) \geq \Phi_{\min}$) because the inter-cluster-distance can be computed in P time.

On the other hand, incorporating new constraints like bounds on $\Phi(\mathcal{C})$ for APOTC or AMSTC, or on the intra-cluster distance $\Theta(\mathcal{C})$, may increase the complexity. The principle of the algorithm remains: guess non-deterministically a clustering, then check if it satisfies the constraints. Hence, the complexity depends on the complexity of the algorithm used as an oracle to check, given a log, a model and a clustering $\mathcal{C}$, if $\mathcal{C}$ satisfies the constraints. Precisely, if there exists such an oracle algorithm in some complexity class A, then the

l-bounded trace clustering problem is in NP^{A}. For instance, for APOTC, checking if $\Phi(\mathcal{C}) \geq \Phi_{\min}$ is in NP; in consequence, the trace clustering problem with such constraint is in $\mathrm{NP}^{\mathrm{NP}}$. We get the same result for constraints on the intra-cluster distance ($\Theta(\mathcal{C}) \geq \Theta_{\min}$) for AMSTC.

4.6 AMSTC SAT Encoding

In this section we show how we approach AMSTC definition with a SAT encoding to get model-based trace variants.

We encode the existence of an AMSTC of m clusters, to get m model-based trace variants, for net N and log L with maximal distance d, as a SAT formula of the equation (4.5). Formally, we check the existence of $\mathcal{C} = \langle\{\mathcal{N}_1 \dots \mathcal{N}_m\}, \chi\rangle$ such as :

$$\bigwedge_{\sigma \in L} \chi(\sigma) \neq nc \Rightarrow dist_{\mathcal{N}}(\sigma, \chi(\sigma)) \leq d \tag{4.5}$$

and optimizes several quality criteria like the number of clustered traces. Those optimization are detailed in sub-Section 4.6.4.

Distance $dist_{\mathcal{N}}$ in Eq.(4.5) is the minimal distance between a trace and a model-based trace variant as subnet. In this thesis, we use the Levenshtein edit distance as presented in Section 2.2.2. This induces Eq.(4.6).

$$\bigwedge_{\sigma \in L} \chi(\sigma) \neq nc \Rightarrow \min_{u_\sigma \in Runs(\chi(\sigma))} \mathcal{L}(\sigma, u_\sigma) \leq d \tag{4.6}$$

Technical Details

The SAT encoding of the Levenshtein edit distance given in Chapter 2 is computationally expensive and induces a lot of boolean variables. In this section, we present an alternative based on a translation of the Levenshtein edit distance to the Hamming edit distance.

We can express Levenshtein distance with Hamming distance ($\mathcal{H}$) and skip actions ($\gg_m$ for model moves and $\gg_l$ for log moves), as :

$$\mathcal{L}(\sigma, u) = \min_{\substack{\sigma^{\gg_m}, u_\sigma^{\gg_l} \\ |\sigma^{\gg_m}| = |u_\sigma^{\gg_l}|}} \mathcal{H}(\sigma^{\gg_m}, u_\sigma^{\gg_l}) \tag{4.7}$$

where $\sigma^{\gg_m}$ ranges over all the words of $(\Sigma \cup \{\gg_m\})^*$ obtained by inserting letters $\gg_m$ in σ and respectively $u_\sigma^{\gg_l}$ ranges over all the words of $(\Sigma \cup \{\gg_l\})^*$. Letters $\gg_m$ and $\gg_l$ represent deletions and insertions.

Technically, we obtain the words $\sigma^{\gg_m}$ for a log trace σ as runs of a Petri net $((N_\sigma)^{\gg_m})$, built from σ as illustrated in Fig. 4.6, which contains a transition $\gg_m$ for model moves.

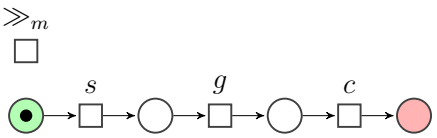


Figure 4.6: The net $((N_\sigma)^{\gg_m})$ Used to Produce the Words $\sigma^{\gg_m}$ for a Log Trace $\sigma = \langle s, g, c \rangle$.

Similarly, the words $u_\sigma^{\gg_l}$ are obtained as runs of a modified subnet, $\chi(\sigma)^{\gg_l}$, which has an additional transition $\gg_l$ for log moves.

Hence, finding $\mathcal{C} = \langle \{\mathcal{N}_1 \dots \mathcal{N}_m\}, \chi \rangle$ satisfying (4.6) amounts to finding the corresponding $\mathcal{C} = \langle \{\mathcal{N}_1 \dots \mathcal{N}_m\}, \chi, (\sigma^{\gg_m})_{\sigma \in L}, (u_\sigma^{\gg_l})_{\sigma \in L} \rangle$ satisfying

$$\bigwedge_{\sigma \in L} \begin{cases} \sigma^{\gg_m} \in Runs((N_\sigma)^{\gg_m}) & (\Phi_1) \\ u_\sigma^{\gg_l} \in Runs(\chi(\sigma)^{\gg_l}) & (\Phi_2) \\ \chi(\sigma) \neq nc \Rightarrow \mathcal{H}(\sigma^{\gg_m}, u_\sigma^{\gg_l}) \leq d & (\Phi_3) \end{cases} \tag{4.8}$$

The SAT encoding of Eq.(4.8) is a conjunction of three building blocks that are detailed in the next subsections.

4.6.1 SAT Encoding of Log Traces

The SAT encoding of log traces gives Φ_1 of Eq.(4.8). Log traces are encoded as sequential Petri nets noted N_σ to deal with model moves. The nets of traces contain an isolated $\gg_m$ transition that can fire at any instant. Moreover, a $\gg_w$ transition, for *"wait"*, is added at the end of the sequential nets to adapt the different sizes of traces. Fig. 4.6 shows the net of trace $\langle s, g, c \rangle$.

SAT encoding of Petri nets has been recalled in Section. 2.3.2 and require two kinds of boolean variables : transition firing and marking representations. We then declare $\tau^{\mathcal{L}}$ and $m^{\mathcal{L}}$ variables for the nets of log traces contained in $|L|$ and defined over alphabet Σ :

- $\tau^{\mathcal{L}}_{\sigma,i,a}$, $for\ \sigma \in L,\ i = 1, \dots, n\ and\ a \in \Sigma$ with n the limited size of run. Those boolean variables indicate that a transition of net of trace σ labeled by a fires at instant i. $\mathcal{L}$ allows one to differentiate net of log trace and the initial process model that also have τ variables.

- $m^{\mathcal{L}}_{\sigma,i,p}$, $for\ \sigma \in L, i = 0, \dots, n\ and\ p \in P_\sigma$ where P_σ is the set of places of net of trace σ. Those boolean variables represent the marking of the net of trace at instant i.

Then, any net of trace $N_\sigma = \langle P_\sigma, T_\sigma, F_\sigma, m_{\sigma_0}, m_{\sigma_f}, \Sigma, \Lambda \rangle$ has the SAT clauses presented in the Background Section (Section 2).

4.6.2 SAT encoding of Model Runs

The SAT encoding of model runs is Φ_2 of Eq.(4.8). Now that nets of traces are encoded, we want to encode alignment between them and the process model which is also a Petri net. Any trace σ is aligned to the model and requires its own run of the model noted u_σ. We then encode in formula (Φ_2) $|L|$ times the process model with the following boolean variables :

- $\tau^{\mathcal{M}}_{\sigma,i,a},\ for\ \sigma \in L,\ i = 1,\ldots,n\ and\ a \in \Sigma$ with n the limited size of run. Those variables indicate that model run of trace σ fires transition labeled by a at instant i.

- $m^{\mathcal{M}}_{\sigma,i,p},\ for\ \sigma \in L, i = 0,\ldots,n\ and\ p \in P$ where P is the set of places of the process model. Those variables represent the required marking of the process model to get run u_σ.

The runs of the model follow the exact same Petri net encoding as the net of traces. Moreover, similarly to nets of traces, we added an isolated transition $\gg_l$ to represent log moves and a *wait* transition to deal with different sizes of traces.

Notice that due to the use of the SAT encoding of Petri nets, the model-based trace variants are forced to be sound and reach the final marking of the initial process model.

4.6.3 SAT Encoding of Variants

The SAT encoding of the variants fills Φ_3 of Eq.(4.8). By using AMSTC, model-based trace variants are subnets $\mathcal{N}_1 \ldots \mathcal{N}_m$ ($m \in \mathbb{N}$). They are defined by transitions of model runs u_σ used to align clustered traces σ of L. This is defined with the conjunction (Φ_3) of (4.8) that we recall:

$$(\Phi_3): \quad \bigwedge_{\sigma \in L} \quad \chi(\sigma) \neq nc \Rightarrow \mathcal{H}(\sigma^{\gg_m}, u_\sigma^{\gg_l}) \leq d \tag{4.9}$$

In this subsection, we first present how the distances are encoded. Then, we detail the implication that incorporate clustered traces.

SAT Encoding of Distances

Aligning nets of traces and runs is obtained by computing the number of differences between fired transitions. We introduce $\delta^{\mathcal{M}}_{\sigma,i}$ and $\delta^{\mathcal{L}}_{\sigma,i} with\ \sigma \in L$ and $i = 1,\ldots,n$ boolean variables

that represents model and log moves.

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n}\bigwedge_{\substack{a\in\Sigma\\ a\neq\gg_l\\ a\neq\gg_m}} (\tau^{\mathcal{L}}_{\sigma,i,a} \wedge \neg\tau^{\mathcal{M}}_{\sigma,i,a}) \Leftrightarrow (\delta^{\mathcal{M}}_{\sigma,i} \wedge \delta^{\mathcal{L}}_{\sigma,i}) \tag{4.10}$$

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n} \tau^{\mathcal{M}}_{\sigma,i,\gg_l} \Leftrightarrow \delta^{\mathcal{L}}_{\sigma,i} \tag{4.11}$$

$$\bigwedge_{\sigma\in L}\bigwedge_{i=1}^{n}\bigwedge_{a\in\Sigma} \tau^{\mathcal{L}}_{\sigma,i,\gg_m} \Leftrightarrow \delta^{\mathcal{M}}_{\sigma,i} \tag{4.12}$$

Axiom (4.10) forces two $\delta_{\sigma,i}$ variables to be `true` when two different activities are aligned which implies a distance to 2, this is equivalent of alignment cost. Indeed, in term of alignment, this situation is represented by a model and a log moves and costs 2.

The maximal distance d given in Equation (4.5) is implemented as *at_most_k* constraints [68], i.e., the number of variables δ to `true` is limited by d:

$$\bigwedge_{\sigma\in L} at\_most\_k(\sum_{i=1}^{n}\sum_{\Delta\in\{\mathcal{L},\mathcal{M}\}} \delta^{\Delta}_{\sigma,i},\ d) \tag{4.13}$$

SAT Encoding of Clustered Traces

Every trace is either clustered in one of the m clusters and attached to a model-based trace variants either associated to the group entitled `nc`, for non-clustered traces. This is encoded with the following variables.

- InC_σ, for $\sigma \in L$ boolean variables that are `true` where trace σ is clustered.
- $\chi_{\sigma,k}$ for $\sigma \in L$ and $k = 0, \ldots, m$ boolean variables that encode which trace is in which cluster.

We then describe trace-cluster associations with the next SAT constraint :

$$\bigwedge_{\sigma\in L} InC_\sigma \quad \Leftrightarrow \quad \bigvee_{k1=0}^{m} (\chi_{\sigma,k1} \bigwedge_{\substack{k2=0\\ k2\neq k1}}^{m} \neg\chi_{\sigma,k2}) \tag{4.14}$$

If a trace is clustered, i.e. InC variable is `true`, transitions of its corresponding runs belong to the model-based trace variant of its cluster. We declare boolean variables that encode which transition belongs to which model-based trace variant.

- $c_{k,t}$ for $t \in T$ and $k = 0, \ldots, m$ with m the number of clusters. Those variables are `true` if transition t is in model-based trace variant k.

Equation 4.15 then describes model-based trace variant-transition associations.

$$\bigwedge_{\sigma \in L} InC_\sigma \Rightarrow (\bigwedge_{i=0}^{n} \bigwedge_{a \in \Sigma} \tau_{\sigma,i,a}^{\mathcal{M}} \Rightarrow \bigvee_{\substack{t \in T \\ \Lambda(t)=a}} \bigvee_{k=0}^{m} (\chi_{\sigma,k} \Rightarrow c_{k,t})) \tag{4.15}$$

Conjunction of expressions (4.10 to 4.15) forms (Φ_3).

4.6.4 Optimization Criteria for AMSTC

The presented SAT formula accepts a large set of solutions. However, to get optimal model-based trace variants, we add three optimization criteria:

- Number of clustered traces should be maximized, i.e., number of non-clustered traces should be minimized.
- Inter-cluster distance, i.e., the distance between model-based trace variant, should be maximized.
- Distances between the traces and the model-based trace variant should be minimized.

This problem of optimization is then a MaxSAT problem that uses the following weighted clauses.

Minimization of Non-Clustered Traces

First, we optimize the number of clustered traces by maximizing InC variables to `true` with the following MaxSAT formula :

$$\sum_{\sigma \in L} InC_\sigma * W1 \qquad \text{where } W1 \text{ is a positive weight} \tag{4.16}$$

Inter-cluster Distance Maximization

To maximize the inter-cluster distance, we use the heuristic that inter-cluster distance is optimal when the number of common transitions between two model-based trace variants is minimized. We then introduce new boolean variables:

- $Common_{k1,k2,t}$ for $k1, k2 \in \{0, \ldots, m\}$, $k1 \neq k2$ and $t \in T$, are boolean variables describing common transitions between centroids of two clusters.

The minimization is found with the following MaxSAT problem where the idea is to set as many as possible $Common_{k1,k2,t}$ variables to `false` which reduces the number of common transitions between model-based trace variants.

$$\sum_{k1=0}^{m} \sum_{\substack{k2=0 \\ k1 \neq k2}}^{m} \sum_{t \in T} \neg Common_{k1,k2,t} * W2 \qquad \text{where } W2 \text{ is a positive weight} \tag{4.17}$$

Minimization of Distances

Finally the minimization of differences can be encoded by the following MaxSAT clauses:

$$\sum_{i=1}^{n} \sum_{\sigma \in L} \sum_{\Delta \in \{\mathcal{L}, \mathcal{M}\}} \neg \delta_{\sigma,i}^{\Delta} * W3 \qquad \text{where } W3 \text{ is a positive weight} \quad (4.18)$$

Weights and Peculiarities in Implementation

This large MaxSAT formula is implemented in DarkSider and da4py presented in Section 1.4.4. The two softwares use SAT solvers to get optimal solutions and return centroids and their associated traces.

Parameter $maxD$ is the maximal distance allowed between a centroid and a trace which was given by the at_most_k constraint of Eq.(4.13). Similarly, in the implementation, the number of transitions per cluster is limited by a parameter, noted $maxCSize$.

We define the following priorities and the implications on the weights:

1. First, traces should be clustered: $W1 = maxD * W3 + maxCSize * W2$
2. Then, number of common transitions should be limited: $W2 = maxD * W3$
3. Finally, distances should be reduced: $W3$

4.7 Sampling Algorithm to Deal with Large Logs

Like for multi-alignment, the SAT formula of the AMSTC algorithm encodes all the traces which requires a lot of memory in practice. We propose a sampling algorithm that helps one to deal with large logs.

4.7.1 Main Sampling Idea

To reduce the formula of the AMSTC algorithm, we propose Alg. 3, a sampling method that calls the AMSTC algorithm only on samples. The AMSTC returns a set of subnets which are the model-based trace variants and their list of clustered traces. Then every trace of the entire log is aligned to each of the discovered variants and added to corresponding cluster if the alignment is sufficiently good. Then we iterate over the remaining traces to cluster with new model-based trace variants.

One can limit the number of model-based trace variants per loop, which also reduces the size of the SAT formula. Alg. 3 takes as input a model N, a log L, a sample size $sampleSize$ to randomly select a sample of the entire log, a counter to stop the research

Algorithm 3: AMSTC Sampling Algorithm

Input : N, L, $sampleSize$, m, $maxCSize$, $maxD$, $maxTrials$

```
1  Clusters = {}
2  counter = 0
3  while |L| > 0 and counter < maxTrials do
4      sublog = randomSampling(L, sampleSize)
5      clustering = AMSTC(N, sublog, m, maxCSize, maxD)
6      if clustering is ∅ then
7          counter = counter + 1
8      else
9          LogAlignToCluster(clustering, L, maxD, Clusters)        ▷ below
10         counter = 0
11 if L > 0 then
12     Clusters[nc] = L                                  ▷ non-clustered traces
```

Output: Clusters : {modelBasedTraceVariant : clusteredTraces}

```
13 Function LogAlignToCluster(clustering, L, maxD, Clusters):
14     foreach cluster in clustering do
15         modelBasedTraceVariant = cluster.getVariant()
16         foreach l in L do
17             if alignmentCost(l, modelBasedTraceVariant) < maxD then
18                 Clusters[cluster].add(l)
19                 L.remove(l)
```

of model-based trace variants when samples cannot be aligned anymore and parameters of the AMSTC algorithm, i.e. the number of model-based trace variants m, the maximal number of transitions per cluster $maxCSize$ and the maximal distance between the traces and their variants $maxD$.

We give the schematization of the algorithm for more readability in Fig. 4.7.

4.7.2 Reducing Alignment Use with Casual Edit Distance between Traces

Alignment is much more expensive than edit distance between words. To limit the use of alignment, we propose another version that first uses edit distance between the clustered traces and the rest of the log. This heuristic allows one to considerably reduce the log before doing alignments.

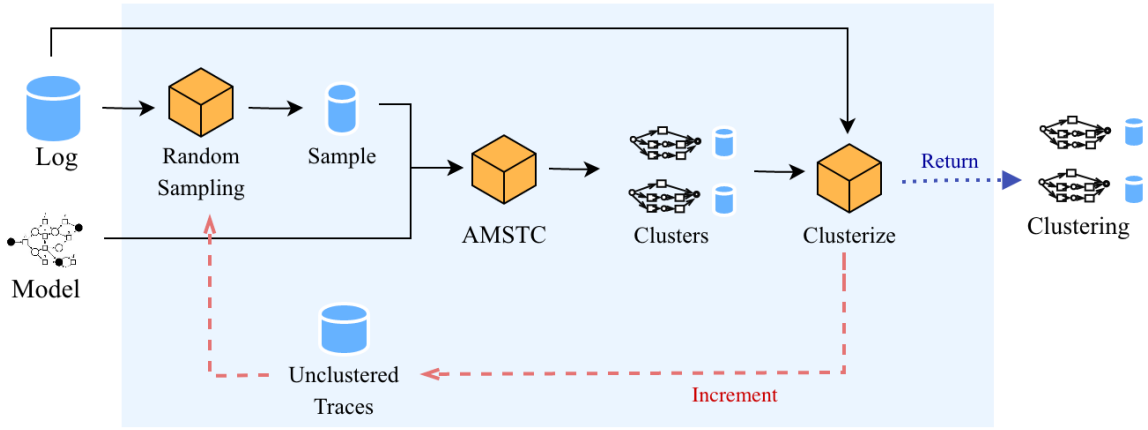


Figure 4.7: Schematization of the AMSTC Samping Algorithm

As any trace is a maximal distance $maxD$ to its model-based trace variant, novel clustered trace is then at maximal distance to its centroid to $2 * maxD$ as the maximal distance between clustered traces and unclustered traces is also $maxD$. One can also consider introducing another input for this purpose, thus raising the number of clustered traces.

4.7.3 Memoization of Alignment Costs

Typically, in real life logs, traces corresponding to frequent behaviors occur many times in the log. This suggests an easy way to improve the efficiency of Alg. 3: it suffices to memoize the calls to the function `alignmentCost` which takes most of the computation time. Tab. 4.5e in the experiments section 4.8 shows the spectacular improvement obtained with this technique.

4.7.4 Statistical Confidence for Sampling

To ensure our random sampling approach, we provide two statistical confidence thresholds.

Probability of Missing Clusterizable Traces

We focus on the situations where Alg. 3 stops before clustering all the log traces which are sufficiently close to a run of the model (i.e. at distance $\leq maxD$), that we call clusterizable traces.

We quantify this probability as a function of the proportion of unclustered traces which are clusterizable. Let p be this proportion when the algorithm starts a series of iterations in order to find a nonempty clustering. As long as the clusterings fail, the unclustered traces remain the same and the proportion p does not change.

Algorithm 4: Reducing Alignment Use of Algorithm 3 (lines 13 to 19)

```
1  Function LogAlignToCluster(clustering, L, maxD, Clusters):
2      foreach cluster in clustering do
3          clusteredTraces = cluster.getClusterTraces()
4          modelBasedTraceVariant = cluster.getVariants()
5          foreach l in L do
6              foreach trace in clusteredTraces do
7                  if editDistance(l, trace) < maxD then
8                      Clusters[cluster].add(l)
9                      L.remove(l)
10             if l is still unclustered then
11                 if alignmentCost(l, modelBasedTraceVariant) < maxD then
12                     Clusters[cluster].add(l)
13                     L.remove(l)
```

Now, a clustering fails precisely when no clusterizable trace is selected in the sample. The probability of this is $(1-p)^{sampleSize}$ (the sampled traces are selected independently one from the other).

Finally, if the algorithm starts a series of clusterings from a state where the proportion of unclustered traces which are clusterizable is p, the probability that it fails $maxTrials$ times to cluster traces (and then stops), is $(1-p)^{sampleSize \times maxTrials}$.

For example, assume 5% of the unclustered traces are clusterizable, the probability that the algorithm fails to detect them after 2 trials with $sampleSize = 10$, is $0.95^{2\times10} \approx 0,36$. Alter 10 trials, the probability drops to $0.95^{10\times10} \approx 0,006$.

If no clusterizable trace remains, i.e. $p = 0$, the trials fail (and then the algorithm terminates) with probability 1.

Wilson score interval

Equivalently to [12], the number of trials in the sampling method can be assessed with a statistical method. Wilson score interval [141] gives a lower bound $\mathcal{L}_b$ and a upper bound $\mathcal{U}_b$ of probability δ to get a probability p of success on a sample of size n with a confidence α:

$$\frac{p + \frac{z^2}{2n} - z * \sqrt{\frac{p*(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \leq \delta \leq \frac{p + \frac{z^2}{2n} + z * \sqrt{\frac{p*(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{4.19}$$

where z is the $1 - \frac{\alpha}{2}$ quantile of a standard normal distribution corresponding to the target error rate α.

In our sampling method, we are looking for the number of trials n such as the probability

p to get success with our $AMSTC$ function, i.e., the sample provide new model-based trace variants, is bounded by δ with a confidence α. In other words, the probability to get new model-based trace variants with a maximal bound δ corresponds to the null hypothesis, i.e., $p = 0$, of the statistical interval. In our approach, we are looking for the minimal number of sample $\mathcal{N}_{min}$ that is sufficient for a given maximal bound δ and a confidence α. From the right side of Wilson score intervals (4.19), we obtain:

$$\mathcal{N}_{min} \geq \frac{z^2 * (1 - \delta)}{\delta} \tag{4.20}$$

To illustrate this inequality we give an example inspired from [12]. We want to find the minimum sample size required to be confident at 0.99 that a novel trial of $AMSTC$ would not give new model-based trace variants with a lower bound probability of 0.95. Then, we set the confidence α to 0.01 which implies $z = 2.58$ from the $1 - \frac{\alpha}{2}$ one-side quantile of the standard normal distribution. The lower bound probability that the novel trial of getting new clusters will fail corresponds to the upper bound to get a success, i.e., to get clusters. Then $\delta = 0.05$. From equation (4.20), we found that the minimal sample size is 127 with Wilson score interval.

4.8 Experiments

In this section, we present our different AMSTC implementations and show a set of experiments from small artificial logs to large real-life logs. As usual, some traces corresponding to frequent behavior occur many times in the log. We write *number of classical trace variants* for the number of different log traces in a log, in reference to previous works on trace variants like [115]. The aim of the present paper is precisely to reduce this number of trace variants using our richer notion of trace variants obtained as centroids of trace clusters.

Implementation of the AMSTC method exists in two tools : DarkSider and da4py. The sampling methods and additive heuristics have been developed in da4py only. All the experiments of this paper have been run with da4py on a virtual machine with 12 CPU Intel Xeon 2.67GHz and 50GB RAM.

4.8.1 Event Logs

Experiments have been done on a set of 7 different logs shown in Tab. 4.4 from 9 to 41353 traces. First log has been presented in [19] to show how model-based clustering helps one to group traces and extracts deviated behaviors. Log presented in section 4.1 is also an artificial log for this purpose. All the other logs have been introduced in other context. We used 3 real-life logs of BPI challenges.

AMSTC algorithm extracts model-based trace variants that required a model as input. To show a large variety of different cases, we have done our experiments on models of literature that have been created in different ways for the respective logs. Tab. 4.4 indicates

Log	$\|L\|$	Number of Classical Trace Variants	$\|\Sigma\|$	$\forall_{\sigma \in L} max(\|\sigma\|)$	Model Discovery Method	$\|T\|$	$\|P\|$
Artificial $L1$ of [19]	9	9	7	7	Hand written model [1]	8	6
Artificial log described in Sec 4.1	500	411	40	36	Hand written model [1]	90	92
Artificial $L1$ of [112]	500	453	37	36	PLG2 tool	39	40
LoanA of [135]	500	100	16	37	Hand written model [2]	17	14
$BPI'2013_{cp}$	1487	183	7	35	Heuristic Miner [3]	25	18
$BPI'2013_{inc}$	7554	1511	13	123	Split Miner [3]	15	11
$BPI'2014_{f}$	41353	14948	9	167	Split Miner [3]	24	16

[1] Available at https://github.com/BoltMaud/da4py/examples
[2] Available at doi:10.4121/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c
[3] From [8], available at https://doi.org/10.6084/m9.figshare.6376592.v1

Table 4.4: Event Logs Statistics and Used Discovered Models

which design method have been used. Complexity of AMSTC depends on the size of the event logs and size of the models which are detailed in the table.

Notice that models with choices, loops and concurrency have been preferred than linear models or mostly concurrent patterns. As our method extracts sound subnets, fully concurrent models cannot be divided in several subnets.

4.8.2 Qualitative Experiments

$\langle s, c, g \rangle$
$\langle s, c, g, d \rangle$
$\langle s, f, b, a \rangle$
$\langle s, f, f, a \rangle$
$\langle s, b, f, a \rangle$
$\langle s, g, f, d, d \rangle$
$\langle s, g, f, d, d, d, d \rangle$
$\langle g, c, f, s, d, d \rangle$
$\langle s, d, d, d \rangle$

Figure 4.8: Log $L2$

This section aims at comparing complete AMSTC and the sampling method outputs with different parameters. We used the log $L2$ given in [19] and its hand written process

model which is small enough to be fully computed by the entire algorithm. The log contains 9 traces of maximal length to 7 and the model has 14 nodes (see Fig. 3.1a).

Experiments have been done with a size of run to 7 and a maximal of trials to 2. Tab. 4.5 gives the results of the experiments. Each sub-table is an improvement of the previous one except table 4.5f which aims at showing consequence of the number of classical trace variants. Every line is an experiment of a specific setting and has been run 10 times. For descriptive results, like the number of clusters and traces, the most returned results are shown. Runtimes are means for the experiments of the selected results. We now give an analysis of the tables.

$\|L\|$	Number of Classical Trace Variants	Sample Size	Maximal Distance (Number of Moves)	Number of Clusters per *AMSTC*	Number of Clusters	Traces Per Cluster			Unclustered Traces	Runtime (secs)	Alignment Runtime (secs)
						Min	Max	Avg			
9	9	/	0	3	3	2	2	2	3	1.30	/
9	9	/	0	2	2	2	2	2	5	1.41	/
9	9	/	2	3	2	3	5	4	1	11.63	/

(a) Complete AMSTC on a small log ($L2$)

90 000	9	10	0	2	3	20 000	20 000	20 000	30 000	1503.98	14095.16
90 000	9	10	2	2	2	30 000	50 000	40 000	10 000	984.90	975.45

(b) Sampling Method on a large Log ($L2 * 10\ 000$)

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	595.37	587.83
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	473.42	465.52

(c) Clustering Effects on Specific Distribution of Traces of $L2$ in large log

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	146.83	133.40
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	151.51	142.65

(d) Edit Distance Heuristic to reduce Alignment Runtime

90 000	9	10	0	2	3	2 000	83 000	29 000	3 000	7.36	0.16
90 000	9	10	2	2	2	3 000	86 000	44 500	1 000	8.45	0.07

(e) Memoization of Alignment Costs

90 000	255	10	0	2	3	18 495	18 866	18 731	33 805	17.95	8.29
90 000	255	10	2	2	3	9 821	50 510	29 825	525	15.88	4.30
90 000	12 460	10	0	2	4	631	2 874	1 601	83593	962.82	935.65
90 000	12 460	10	2	2	3	15 366	35 964	22 855	21 433	572.87	560.10

(f) Clustering Effects on Noisy Log (Raising the Number of Classical Trace Variants)

Table 4.5: Comparison of AMSTC Results for Different Parameters on Log $L2$ of [19].

In sub-table 4.5a, we see that, given different distances and number of clusters, results differ. Raising the distance between the traces and the centroids allows to cluster more traces. However, a good distance threshold aims at partitioning the traces in more specific

clusters and then get specific model-based trace variants.

In sub-table 4.5b, sampling method outputs are presented. For the exact same distribution of the traces, we proportionally get the same results of the complete AMSTC method. Then the exact same model-based trace variants are extracted.

Sub-table 4.5c aims at showing that our method can deal with strange trace variant frequency. In this experiment, the second trace of $L2$ have been duplicated 82000 times while the other traces appear 1000 times each. We can see that the size of the clusters are indeed very different. However, notice that the number of clusters corresponds to the previous experiments. The same model-based trace variants have been extracted.

Sub-tables 4.5d and 4.5e contain heuristic improvements in term of runtimes.

Finally, we have added noise in log to raise the number of trace variants which is used by the heuristics. The number of clusters is then different which is expected.

Experiment Conclusion

From this experiments, we see that the AMSTC method helps one to extract good model-based trace variants. The sampling method efficiently works for large logs and specific distributions of trace variants. For noisy logs, more model-based trace variants are extracted. To reduce the number of model-based trace variants and un-clustered traces, one can change the maximal distance between trace and variants.

4.8.3 Quantitative Experiments

In this section, we present experimentation of the set of different logs presented in section 4.8.1. Settings presented in the table have been chosen by the author after some tests. The tests consisted on evaluating the distance between the traces and the models, assessing the minimal size of the run depending on the traces and loop behaviors and counting an approximate maximal number of transitions per cluster. The sampling size have been chosen in a way that runtime is optimized. The number of clusters per loop was set to 2 to reduce the formula size. Finally, we set the number of trials of the sampling algorithm to 5 sequential fails.

Each experiment have been run 10 times except the last one because of long runtimes. Due to trace variants and the use of causal edit distance between traces, outputs of the same experiment are different. In Tab. 4.6 we show examples of clustering results that have returned the least unclustered traces. Notice that the runtimes are much larger because of the number of activities and trace variants in logs. Moreover, the models are also much larger than in the previous experiments.

Log	Number of Classical Trace Variants	Sample Size	Size of run	Maximal Number of Transition per Cluster	Maximal Distance (Number of Moves)	Number of Clusters	Number of Traces Per Cluster			Unclustered Traces	Runtime (secs)	Alignment Runtime (secs)
							Min	Max	Avg			
Artificial $L1$ of [19]	9	5	5	5	0	3	2	2	2	3	4.02	0.13
Artificial log presented in Sec 4.1	411	5	15	9	0	12	22	54	42	0	2135.84	27.96
Artificial $L1$ of [112]	453	5	15	15	4	4	19	145	62	252	6681.47	666.23
LoanA of [23]	100	10	20	14	1	10	10	60	25	250	409.32	64.31
$BPI'2013_{cp}$	183	10	20	9	1	3	32	1121	451	134	245.29	42.33
$BPI'2013_{inc}$	1511	5	20	11	2	2	204	5981	3092	1369	4091.79	3646.47
$BPI'2014_f$	14948	5	20	15	2	6	257	21909	4344	15289	66709.70	66130.29

Table 4.6: Examples of AMSTC Outputs on a Set of 7 Logs

We can see that our running example presented in Sec 4.1 perfectly associates every trace of the log to a model-based trace variants. For 500 traces and 411 classical trace variants, the AMSTC method finds 12 model-based trace variants. Those subnet instances are then a good way to analyze the log traces separately.

We see that for more noisy logs contained in real-life data and a small distance between trace and variants, we are able to cluster a good number of traces in a very small number of clusters. We see from the number of classical trace variants in the second column that the method is indeed able to group them in our more general model-based trace variants.

Experiment Conclusion

In those experiments, we highlight how our method is able to extract a small number of model-based trace variants of real-life logs compared to the number of classical trace variants. We see that still a lot of traces remain unclustered but this is due to the distance between variants and traces which is intentionally small. The extracted model-based trace variants are well fitting to the clustered traces (maximal distance is always lower than 4 in Tab. 4.6).

4.8.4 Case Study

To present the value of our clustering and the discovered model-based trace variants, we present a case study on a real-life log and different process models. Then, we compare our work to the *Super-instances* from [39] which also aims at representing group of traces and are also found by using a clustering approach.

Event Log

We employed the real-life log from the Business Process Management Challenge of 2013 about the *closed problems* of Volvo management system. The event log contains 1487 log traces, 183 classical trace variants, 7 activity steps (taking in account activity name and progress) and 4 main activity names. Tab. 4.7 shows the frequency traces containing the activity names. We see that some activities, like `Unmatched`, are much less frequent than other, like `Completed`. We also notice that activity `Queued` appears in many classical trace variants but those trace variants are not very frequent in the log. This simple table will help to get good intuition in the understanding of the results of the model-based variants and the super-instances from [39].

Activity Label	Frequency of Traces Containing the Activity Label	Frequency of Classical Trace Variants Containing the Activity Label
Accepted	1.00	0.99
Completed	1.00	1.00
Queued	0.36	0.84
Unmatched	0.01	0.04

Table 4.7: Frequency of Traces and Classical Variants Containing the Different Activities

How Model Quality Impacts Model-based Trace Variants

By using the classical trace variants, i.e, the number of unique sequences, one obtains 183 instances for BPIC 2013 closed problems event log. For human analysis and business aspect, this number of instances is too large to be understood. Our AMSTC method helps one to get more representative variants. However, our method is based on an existing model. In Tab. 4.8, we show different clustering results for different model qualities.

Model	Fitness	Maximal Number of Transition per Cluster	Maximal Distance (Number of Moves)	Number of Clusters	Number of Traces Per Cluster			Unclustered Traces
					Min	Max	Avg	
Split Miner	0.98	10	0	4	1	917	251	485
			2	2	59	1377	718	51
Heuristic Miner	0.94	12	0	2	3	917	460	567
			2	2	17	1391	704	79
Inductive Miner	0.82	7	0	1	574	574	574	913
			2	1	1367	1367	1367	120

Models from [8], available at https://doi.org/10.6084/m9.figshare.6376592.v1

Table 4.8: AMSTC on BPIC 2013_{cp} Log and Different Models. Sample size is set to 15 and run size to 20.

We observe that the fitness of the model impacts the number unclustered traces. Moreover, we see that results of the last model of Tab. 4.8 do not give useful information in term of representatives. In fact, this model, learned with the inductive miner, is very simple and contains only a choice and a loop, making hard to split the model in model-based trace variants. Finally, we note that our clusters are very heterogeneous. This aspect can help one to understand the structure of the model by analyzing the model-based trace variants.

Experiment Conclusion

In this experiment, we show how fitness impacts results of the ASMTC methods, thus giving different model-based trace variants. For larger distances to the centroids, one gets less variants but can cluster more traces from the event log. In opposite, for a distance to zero, the number of model-based variants raises but many traces are left unclustered, i.e, do not have a representative.

Comparison of Clusters and Representatives

Our method extracts Model-based Trace Variants along with clustered traces. This aspect motivates us to compare our method to the Super-Instances of [39]. In this work, it also finds clusters from the log traces, and work with the centroids of them which they call *Super-Instances*. We see that both methods claims to get representatives of groups of traces. Moreover, they also use a clustering method, i.e., the K-means algorithm. The algorithm is based on the Euclidean distance between vectors of activities occurrence contained in the traces. The main differences between our approach and [39] are: first, our approach does not need as input the number of clusters, whilst [39] does. Second, [39] tries to obtain a balanced clustering, i.e., a clustering where the size of the computed clusters are as much similar as possible. In contrast, our method only focuses on finding good representatives of each cluster, regardless of its size. These differences are corroborated in the case study found below.

We used the Jaccard index [137] to compare two clusterings, $\mathcal{C}$ and $\mathcal{C}'$, defined by:

$$\mathcal{J}(\mathcal{C}, \mathcal{C}') = \frac{n_{11}}{n_{11} + n_{01} + n_{10}} \tag{4.21}$$

where:
n_{11} is the number of pairs of items clustered together in $\mathcal{C}$ and in $\mathcal{C}'$
n_{10} is the number of pairs of items clustered together in $\mathcal{C}$ but in different clusters in $\mathcal{C}'$
n_{01} is the number of pairs of items in different cluster in $\mathcal{C}$ but in same clusters in $\mathcal{C}'$

In our study, the Jaccard index relates pairs of traces in the two clustering results, we also report how many pairs of traces have been clustered together in both clusterings (column n_{11} in Tab. 4.9), and how many traces were clustered together in a clustering but not in the other one (columns n_{01} and n_{10} in Tab. 4.9). A high value (≤ 1) of the Jaccard index indicates that the clustering are similar. The main of this study is then to study our clustering results of Tab.4.8 and the corresponding outputs of [39].

Cluster Sizes of AMSTC (see Tab. 4.8)			Jaccard Comparison				Cluster Sizes by using method of [39]		
Min	Max	Median	n_{11}	n_{10}	n_{01}	Index	Min	Max	Median
1	917	42	169577	253652	1560	0.399	82	496	212
59	1377	718	550325	398762	32081	0.561	458	978	718
3	917	470	210957	209032	1479	0.501	424	496	460
17	1391	704	562403	404478	7038	0.577	431	977	704
574	574	574	164451	0	0	1.000	574	574	574
1367	1367	1367	933661	0	0	1.000	1367	1367	1367

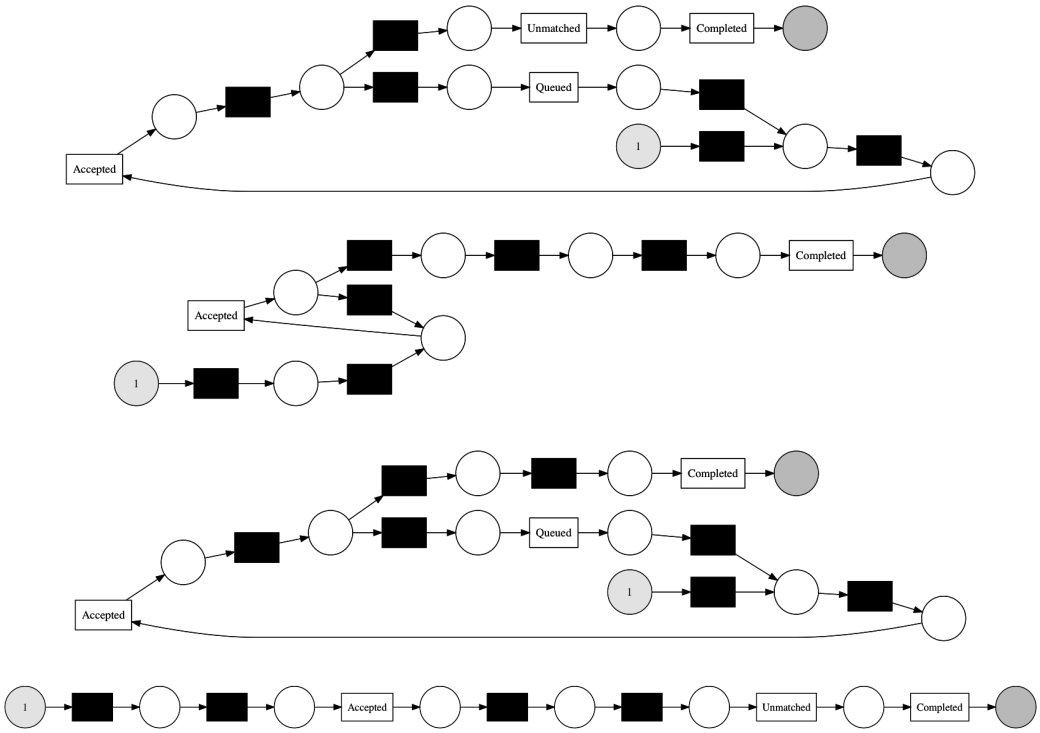
Table 4.9: Cluster Comparison between AMSTC and [39] Results for the Same Traces and Number of Clusters. All the rows of this table have been produced based on the rows of Tab.4.8

.

First, we want to justify about the use of clustered traces only when instructing [39]: in order to compare the clusterings, which aim at grouping traces for their similarities, we claim that considering as a cluster the sets traces not clustered by our method would not give an appropriate comparison, since these unclustered traces are not related. So, to compare the clusters given by the two methods, we ran method of [39] only on the traces that have been clustered in Tab. 4.8.

In opposite of AMSTC, method of [39] requires, as input, the number of clusters. For each ASMTC presented in Tab. 4.8, we launched the method on the clustered traces and used the same number of clusters. For instance, for the first line, we ran the Super-Instance method on the 1002 clustered traces and set the number of clusters to 4. We give in Tab.4.9 an overview of the cluster sizes and the Jaccard Index. The two last rows of the table are not informative as their is only 1 cluster. For the first line, in which 4 clusters have been discovered, we see that the Jaccard Index is under 0.5, meaning that there are more pairs of traces clustered in different clusters by the clustering methods. For the other rows, we see that, the Jaccard index varies from 0.501 to 0.577 which is better than hazard for those lines which have only 2 clusters (see Tab. 4.8 for the number of clusters). Indeed, for 2 clusters, the probability of n_{11}, a pair of traces to be clustered together by both methods, is 1/4, while the addition of the probabilities of n_{10} and n_{01} is 1/2. We see that the Jaccard Index tells us that our method have some similarities.

To conclude this section, we present in Fig. 4.9 the model-based trace variants of our approach and the super-instances of [39] of the first line of Tab.4.9. We remark that two of our representatives contain activity `Queued` and `Unmatched` while the super-instances only have once the activity `Queued`. This is due to the fact that method of [39] uses K-means that tries to get centroids which are means of the occurrences of activities and n-grams of activities. In our method, the model-based trace variants are subnets which can contain choices, thus allowing activities that are not used in all the traces of a cluster. The variant is then more general than a sequence. However, our method can also get sequential net (see Fig. 4.9). Model-based trace variants are a good balance to represent the traces, especially when the distance to the traces is zero, i.e., the variants can replay the traces. This last aspect cannot be induced by using the super-instances.



(a) Model-based Trace Variants

$\langle Accepted, Completed \rangle$
$\langle Accepted, Accepted, Completed \rangle$
$\langle Accepted, Queued, Accepted, Completed \rangle$
$\langle Accepted, Accepted, Accepted, Accepted, Accepted, Completed \rangle$

(b) Super-Instances

Figure 4.9: Representatives of Clusters

Experiment Conclusion

In this comparison, we aimed at showing the differences between the super-instances, which are sequences, and the model-based trace variants. We have seen that the discovered activities slightly differ from a method to another. The AMSTC gives more complex structures, but the resulted representatives provide more information thanks to the semantics of Petri nets. In addition to different structures, the two approaches have completely different underlying search algorithms. Our method tries to get subnets by using alignment between the model and the log traces. In [39], the distance is the Euclidean distance between vectors of occurrences and n-grams, i.e., sub-sequences of traces, and a balanced clustering is obtained if possible. Finally, we want to point out that our method does not require the number of clusters a priori, i.e., our method searches good representatives.

4.9 Conclusion

This chapter enables to cluster log traces by behavior thanks to model-based approach. As process models contain causality, concurrency, choice and loop behavior and can be validated with the conformance checking criteria, we motivate their use as a baseline of clustering approach. By aligning the log traces to the model, we discover the important parts of the model that represent the traces. In this work, we extend the idea of [33] where the discovered centoids are now processes or subnets of the initial model such that concurrency and loops are addressed. All methods use alignment artefact to relate the model to the traces.

We introduced a set of quality criteria to get good clustering results and presented a SAT encoding that outputs the desired clusters. As the SAT encoding to align the entire log requires too much memory, we inaugurated a sampling approach whom we ensure with statistical confidence.

The alignment-based clustering methods return clusters of the log traces along with centroids that we propose to use as model-based variants. We presented some experiments and compared our method to the super-instances defined by [39].

The limit of this work is certainly the sampling approach that dismiss the overall clustering idea because it proceeds on several iterations. The goal of clustering is to group traces for their similarities and highlight the differences between groups. However, by using a sample only, we do not have a overview of all the traces to do the balancing between the quality criteria for a good clustering.

Moreover, we observed in the experiments that some clustering have left many traces unclustered. This results can appear when the model is not much fitting the log and the distance between the centroids and the log traces is too low. One can avoid this by checking the conformance of its process model before any model-based clustering. This idea bridges the gap to the next chapter about anti-alignments.

Chapter 5

Anti-alignments for Measuring Precision and its Interest in Model-based Clustering

Chapter Overview

The present chapter focuses on anti-alignments. In opposite to alignments and multi-alignments, this conformance checking artefact is a run of the process model that is as far as possible to any trace of the given log. The artefact is used for computing precision but also for highlighting model deviations with respect to a log.
The chapter is divided as follows. First, we give the related work for model deviations and precision of process models. In Section 5.2 we give the definitions of anti-alignment and precision. The sub-Section 5.3 provides algorithms for computing both optimums and approximations of anti-alignments. Then, in Section 5.5, we show the impact of using anti-alignment in model-based clustering context. Section 5.6 gives an opening of anti-alignment use in the context of model repair.

5.1 Related Work

We start this chapter by stating the related work that allows to contextualize anti-alignments and model precision whom focus differs from the rest of this thesis. We first present the state-of-the-art methods for precision of process models which is much discussed in the community. Then we give the introduction of anti-alignments.

5.1.1 Precision of Process Models

The work of [104] is one of the first attempts to evaluate the precision of a process model with respect to an event log. It is grounded on comparing relation matrices from the model

and log. Since it requires the full state-space exploration of the process model, it is only applicable to small models.

In [86] deviations are estimated by the number of *escaping arcs*, i.e., runs of process models that deviate from the log. The state-space exploration of this method is bounded by the log behaviors. They provide a precision estimation called *Escaping Edges Precision* (ETC). However, this work does not consider the size of the deviations, i.e., escaping arcs might cause large deviant behaviors.

Recently, an effort to consolidate a set of desired properties for precision metrics has been proposed [109]. Five axioms are described that establish different features of a precision metric $prec(N, L)$. Summarizing, the proposed axioms are:

- $A1$: A precision metric should be a function, i.e. it should be deterministic.
- $A2$: If a process model N_2 allows for more behaviors not seen in a log L than another model N_1 does, then N_2 should have a lower precision than N_1 regarding L:

$$L \subseteq Runs(N_1) \subseteq Runs(N_2) \implies prec(N_1, L) \geq prec(N_2, L)$$

- $A3$: Let N_1 be a model that allows for the behavior seen in a log L, and at the same time its behavior is properly included in a model N_2 whose language is Σ^*[1] (called a flower model). Then the precision of N_1 on L should be strictly greater than the one for N_2.
- $A4$: The precision of a log on two language equivalent models should be equal:

$$Runs(N_1) = Runs(N_2) \implies prec(N_1, L) = prec(N_2, L)$$

- $A5$: Adding fitting traces to a fitting log can only increase the precision of a given model with respect to the log:

$$L_1 \subseteq L_2 \subseteq Runs(N) \implies prec(N, L_1) \leq prec(N, L_2)$$

The two works presented above fail at satisfying these important axioms. We now turn the focus to recent proposals that, as the case for anti-alignment based precision, do satisfy the reference axioms for precision.

Work of [91] transforms recorded and modeled behaviors into abstraction called *directly follows automaton*, and compares their languages. By introducing the notion of *language quotients* according to a measure, a precision metric can be defined, which is the ratio of common sequences between the log and the model to the total model runs. To deal with infinite languages, they use the *topological entropy of languages*, detailed in [27]. However this technique is very strict when any sequence is shared by the log and the model. In [70], they add *skips* actions to make the approach more flexible.

[1] Actually, [109] writes "$Runs(N_1) \subset \mathcal{P}(\Sigma^*)$", with $\mathcal{P}$ for power set, but we believe this is a mistake.

Finally, the work of [8] is based on *Markovian abstractions*. By comparing the k-th order Markovian abstraction of a process model against the respective one of an event log, a precision metric can be obtained. Both works above represent an interesting attempt to estimate precision, but due to being grounded in sometimes aggressive abstractions, they fail in generating concrete insights pinpointing the real deviations. In contrast, anti-alignments are not based on abstraction but instead in concrete model runs, that may serve as a concrete explanation for repairing precision problems in a process model.

5.1.2 Introduction of Anti-alignments

Anti-alignments have been introduced by [31] as the *dark side* of process models. The same year, the authors present in [129] how those conformance artefacts can be used for measuring *precision* but also *generalization*, i.e., two fundamental metrics still in elaboration in process mining [118]. Anti-alignments have been proposed for both the Hamming distance and the Levenshtein distance [30]. Anti-alignment based precision metrics satisfy the necessary axioms for a precision metric [109]. Levenshtein distance is preferred, since it provides a more precise characterization of a deviation. Anti-alignments can be seen as the counterpart of alignments [4]. While many optimizations exist for alignment computation [73, 101, 113], none of them can be adapted to compute anti-alignments since for the latter, the whole log and not only one single trace needs to be considered.

5.2 Definitions

This section completes the aforementioned previous works about anti-alignments. In. [30], which belongs to the set of contributions of this thesis (numerated **C5** in Section. 1.4.3), we formalized and improved anti-alignment computation. Then, the following content is a consolidation of the state-of-the-art for anti-alignments. Entire novelty starts from Section. 5.3 where we propose novel algorithms, experiments and applications.

5.2.1 Anti-alignments

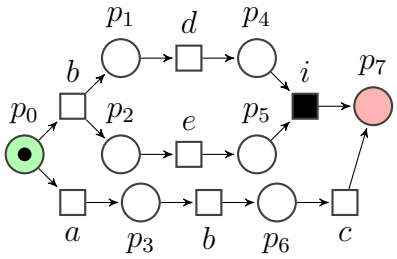
We call *anti-alignments* the most deviant full runs of process models.

Definition 24 (Anti-alignment)**.** *Given a log L and a model N, an (optimal)* anti-alignment *is a full run $u \in Runs(N)$ such that it maximizes the minimal distance $\min_{\sigma \in L} dist(\sigma, \gamma)$ to the log:*

$$\max_{u \in Runs(N)} \min_{\sigma \in L} dist(\sigma, u) \tag{5.1}$$

where dist is a distance between sequences.

Like the other conformance checking artefacts, the preferred distance is the Levenshtein edit distance.



(a) Model $M3$

$\{\langle b, c\rangle,$
$\langle b, d, f\rangle\}$

(b) Log $L3$

Figure 5.1: Drawing Example for Anti-alignments.

Example 5.2.1 (Anti-alignments)**.** *For the process model and the log of Fig. 5.1, full run $\langle b, e, d, \tau\rangle$ is an anti-alignment of minimal distance to any log trace of 2. Observe that run $\langle b, d, e, \tau\rangle$ is also an optimal anti-alignments for this log and model.*

Comparison to Multi-alignments

Like multi-alignments, anti-alignments is a run that involve the entire log at once. However, the complexity of anti-alignments is higher as it implodes to observe all the runs. This it well depicted when the problems are written as 2QBF instances (Quantified Boolean Formula with 2 quantifiers).

Multi-alignments are given by:

$$\exists u \in Runs(N)\ \forall \sigma \in L\ dist(\sigma, u) \leq d \tag{5.2}$$

while anti-alignments are:

$$\exists u \in Runs(N)\ \forall \sigma \in L\ dist(\sigma, u) > d \tag{5.3}$$

which can be transformed into

$$\forall u \in Runs(N)\ \exists \sigma \in L\ dist(\sigma, u) \leq d \tag{5.4}$$

where the quantifier $\forall$ appears first and for the runs of the Petri net.

5.2.2 Anti-alignment-based Precision of Process Models

The main motivation of finding anti-alignments is to estimate precision of process models.

Definition 25 (Anti-Alignment based Precision)**.** *Let L be an event log and N a model. We define precision as follows:*

$$P_{aa}(N, L) \stackrel{\text{def}}{=} 1 - \max_{\gamma \in Runs(N)} \min_{\sigma \in L} \Delta(\gamma, \sigma) \tag{5.5}$$

where $\Delta(\gamma, \sigma) \stackrel{\text{def}}{=} \frac{dist(\gamma,\sigma)}{|\gamma|+|\sigma|}$ *is a normalization of the distance.*

Example 5.2.2 (Precision Computation)**.** *Precision of model N3 for log L3 of Fig. 5.1 is then* $P_{aa}(N3, L3) = 1 - \frac{2}{7} = 0.71$.

Handling Process Models with Loops

Notice that a model with arbitrary long runs (i.e., a process model that contains loops) may cause the formula in Definition 25 to converge to 0. This is a natural artefact of comparing a finite language (the event log), with a possibly infinite language (the process model). Since process models in reality contain loops, an adaptation of the metric is done in this section, so that it can also handle this type of models without penalizing severely the loops.

Definition 26 (Precision for Models with Loops)**.** *Let L be an event log and N a model. We define ϵ-precision as follows:*

$$P^{\epsilon}_{aa}(N, L) \stackrel{\text{def}}{=} 1 - \sup_{\gamma \in Runs(N)} \min_{\sigma \in L} \frac{\Delta(\gamma, \sigma)}{(1+\epsilon)^{|\gamma|}} \tag{5.6}$$

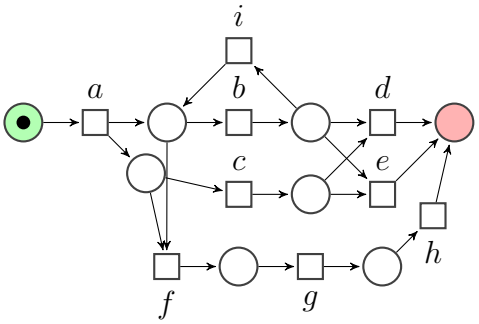
with $\Delta(\gamma, \sigma) = \frac{dist(\gamma,\sigma)}{|\gamma|+|\sigma|}$ *and* $\epsilon \geq 0$.

Informally, the formula computes the anti-alignment that provides maximal distance with any trace in the log, and at the same time tries to minimize its length. The penalization for the length is parameterized over the ϵ.

Technical Details

Admittedly, ϵ is a parameter that should be decided a priori, in practice one can use a particular value to this parameter thorough several instances, without impacting significantly the insights obtained through this metric.

. Observe that $P^0_{aa}(N, L)$ is precisely the precision $P_{aa}(N, L)$ of Definition 25. By making Definition 33 not dependant on a predefined length, it deviates from the log-based precision metrics defined in [31, 129].



(a) Model $M4$ from [5]

$\{\langle a\rangle,$
$\langle a, b, c, d\rangle,$
$\langle a, f, g, h\rangle,$
$\langle a, b, i, b, c, d\rangle\}$

(b) Log $L4$

Figure 5.2: Drawing Example with Loop for Anti-alignments.

Example 5.2.3 (Epsilon Influence)**.** *Let us now consider the model of Figure 5.2a, and the log. Assume that* $\epsilon = 0.05$*. Let* $\gamma_1 = \langle a, c, b, e\rangle$ *a full run which is at least at distance* $\frac{1}{2}$ *to any of the log traces. For* γ_1 *the value of the formula is* $1-\frac{\frac{1}{2}}{(1.05)^4} = 0.589$*. Now, consider* $\gamma_2 = \langle a, c, b, i, b, i, b, i, b, i, b, e\rangle$ *which is at least at distance* $\frac{10}{18}$ *to any of the log traces. For* γ_2 *the value of the formula is* $1 - \frac{\frac{10}{18}}{(1.05)^{12}} = 0.691$*. Since the full run that maximizes the second term of the formula is* γ_1 *which stands as anti-alignment. The precision then is* $P_{aa}^{0.05}(N4, L4) = 0.589$*. If instead,* ϵ *is set to a lower value, e.g.,* $\epsilon = 0.02$*, the corresponding value of the formula for the anti-alignment* $\langle a, c, b, i, b, i, b, i, b, i, b, i, b, i, b, i, b, e\rangle$ *will be the minimal, and therefore it will be selected as the anti-alignment resulting in* $P_{aa}^{0.02}(N4, L4) = 0.533$*.*

5.2.3 Complexity of Precision Computation

By incorporating the ϵ parameter in the definition of precision, now the metric can deal with models containing loops without predefining the length of the anti-alignment. In this section we show that the proposed extension is well-defined and can be computed, and provide some complexity results of the algorithms involved.

Lemma 7. *For every finite model* N*, log* L *and* $\epsilon > 0$*, the supremum in the definition of* P_{aa}^{ϵ} *is reached, i.e. there exists a full run* $\gamma \in Runs(N)$ *such that* $P_{aa}^{\epsilon}(N, L) = 1 - \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}}$ *with* $\Delta(\gamma, \sigma) = \frac{dist(\gamma,\sigma)}{|\gamma|+|\sigma|}$*.*

Proof. Two cases have to be distinguished: if $Runs(N) \subseteq L$, then the supremum equals 0, is obviously reached by any $\gamma \in Runs(N)$, and $P_{aa}^{\epsilon}(N, L) = 1$; otherwise, let $\gamma_0 \in Runs(N)\backslash L$ and let $m \stackrel{\text{def}}{=} \frac{\Delta(\gamma_0, L)}{(1+\epsilon)^{|\gamma_0|}}$; we show that the supremum in the definition of P_{aa}^{ϵ} becomes now a

maximum over a finite set of runs, bounded by a given length n that depends on m and ϵ:

$$\sup_{\gamma \in Runs(N)} \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}} = \max_{\gamma \in Runs(N), |\gamma| \leq n} \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}}, \tag{5.7}$$

with $n \stackrel{\text{def}}{=} \left\lfloor \frac{-\log m}{\log(1+\epsilon)} \right\rfloor$. Indeed, for every γ strictly longer than n, we have $\frac{\Delta(\gamma,L)}{(1+\epsilon)^{|\gamma|}} \leq \frac{1}{(1+\epsilon)^{n+1}} = \exp(-(n+1) \cdot \log(1+\epsilon)) \leq \exp(\log m) = m$, which also shows that $|\gamma_0| \leq n$. Hence γ_0 is considered in our max, and then $\max_{\gamma \in Rus(N), |\gamma| \leq n} \frac{\Delta(\gamma,L)}{(1+\epsilon)^{|\gamma|}} \geq m > \sup_{\gamma \in Runs(N), |\gamma| > n} \frac{\Delta(\gamma,L)}{(1+\epsilon)^{|\gamma|}}$.

□

Lemma 7 gives us the key for an algorithm to compute P^{ϵ}_{aa}.

Algorithm 5: Algorithm for Computing $P^{\epsilon}_{aa}(N, L)$

Input : N, L, ϵ
Output: $P^{\epsilon}_{aa}(N, L)$

```
if Runs(N) ⊈ L then
    select γ0 ∈ Runs(N) \ L
    m ← Δ(γ0,L) / (1+ε)^|γ0|
    Explore the reachability graph of N until depth n ≝ ⌊ -log m / log(1+ε) ⌋
    return 1 − max_{γ∈Runs(N),|γ|≤n} Δ(γ,L) / (1+ε)^|γ|;
else
    return 1                                  ▷ The model has perfect precision
```

The correctness of this algorithm follows directly from Lemma 7. Its complexity resides essentially in the initial test, which corresponds to simply deciding if $P^{\epsilon}_{aa}(N, L) < 1$, whose complexity is given by the following lemma:

Lemma 8. *The problem of deciding, for a finite model N and a log L, if $P^{\epsilon}_{aa}(N, L) < 1$, is equivalent to deciding reachability in Petri nets.*

Proof. We simply observe that $P^{\epsilon}_{aa}(N, L) < 1$ iff $Runs(N) \not\subseteq L$. Deciding this is equivalent to deciding reachability in Petri nets.

□

Technical Details

In practice, one would generally skip the first check and jump directly to the exploration until some depth n, possibly computed for a given threshold m, like the one given by the γ_0 in Alg. 5.

Notice that the algorithm explores less deep (i.e. n is smaller) when m is large (close to 1), i.e. γ_0 is close to the optimal anti-alignment. We can summarize this with the following variation of Alg. 5:

Algorithm 6: Algorithm for Estimating $P^{\epsilon}_{aa}(N, L)$ using a Threshold $0 < m \leq 1$ as Input

Input : N, L, ϵ, m
Output: $P^{\epsilon}_{aa}(N, L)$

1 Explore the reachability graph of N until depth $n \stackrel{\text{def}}{=} \left\lfloor \frac{-\log m}{\log(1+\epsilon)} \right\rfloor$
2 **if** *Exists* $\gamma \in Runs(N) \setminus L$ **then**
3 **return** $P^{\epsilon}_{aa}(N, L) = 1 - \max_{\gamma \in Runs(N), |\gamma| \leq n} \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}}$
4 **else**
5 **return** $P^{\epsilon}_{aa}(N, L) \geq 1 - m$

Lemma 9. *For any fixed $\epsilon > 0$, the problem of deciding, for a finite model N, a log L and a rational constant $m > 0$, if $P^{\epsilon}_{aa}(N, L) < 1 - m$, is NP-complete.*

Proof. The proof is similar to the one of Lemma 7; here, the bound m is given directly, and we have the same equality

$$\sup_{\gamma \in Runs(N)} \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}} = \max_{\gamma \in Runs(N), |\gamma| \leq n} \frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}}, \tag{5.8}$$

with $n \stackrel{\text{def}}{=} \left\lfloor \frac{-\log m}{\log(1+\epsilon)} \right\rfloor$. This means, in order to check that $P^{\epsilon}_{aa}(N, L) < 1 - m$, it suffices to guess a full run γ of length $|\gamma| \leq n$, where n depends linearly on the size of the representation of m (number of bits in the numerator and denominator). Then one can check in polynomial time that $\frac{\Delta(\gamma, L)}{(1+\epsilon)^{|\gamma|}} > m$.

□

5.3 Algorithms for Computing Anti-alignments

The algorithms for computing anti-alignments are variants of the algorithms presented in Chapter 3 for computing alignments and multi-alignments. In this section we give the differences to compute anti-alignments, which also draws the relationships between the conformance checking artefacts. We recall that the two algorithms are :

- **a MinSAT-based algorithm** which encodes the definition as it is for getting optimum results
- **an A*-based algorithm** which uses the discounted edit distance and contains heuristics to get fast results for real-life application

5.3.1 MinSAT Encoding

The SAT problem of finding an anti-alignment can be set as follows: *Does an anti-alignment exist between the log traces L and the model N for a minimal distance d?* The only difference to multi-alignments and alignments is the search of maximizing the minimal distance between the model run and the log traces where the other artefacts aim at minimizing the maximal distance. Then the building block for relating the process model and the traces with the edit distance encoding is the same for the three approaches. We encode:

- process models as described in Section 2.3.2,
- the edit distance like in Section 3.3.1,
- and the relation between the model and the traces equals the one of Section. 3.3.2.

Now the optimization goal, that involves a MinSAT encoding, differs. We recall that for multi-alignments, the minimization of the distance works as follows:

$$\bigwedge_{d} \left(\bigvee_{\sigma} \delta^{\sigma}_{n,|\sigma|,d} \Leftrightarrow \Delta_d \right) \tag{5.9}$$

where $\delta^{\sigma}_{n,|\sigma|,d}$ encodes that trace σ is at least at distance d to the chosen run of the model at instant n, which is the size of the run, and $|\sigma|$, which is the length of the trace. The Δ_d variables define the distances d for which at least one of the traces verifies these distances to the run of the model. The minimization objective for multi-alignment is: $\sum_d 1 \times \Delta_d$.

For anti-alignment, Equation (5.10) becomes:

$$\bigwedge_{d} \left(\bigwedge_{\sigma} \delta^{\sigma}_{n,|\sigma|,d} \Leftrightarrow \Delta_d \right) \tag{5.10}$$

where Δ_d variables encodes that all traces verify a distance d to the run. Then, contrary to multi-alignments, we want the maximum of these variables to **true**. For a MinSAT problem, the goal is to minimize: $\sum_d 1 \times \neg\Delta_d$, i.e., we want the least number of Δ_d variables to *false*.

We observe that the encoding is very easily adapted from multi-alignment to anti-alignment which is one strength of the method.

Formula Reduction

In Section. 3.3.4, we have shown how, relying on the minimization of the δ variables, we can reduce the SAT encoding of the edit distance $\Phi_{\Leftrightarrow}$ for multi-alignments noted $\Phi_{\Leftarrow}$. Similarly, we are able to optimize this encoding for anti-alignments.

The optimization for multi-alignments enables to remove the left side of the equations. In opposite, for anti-alignments, keeping the right parts only give an optimal reduction of

the formula thanks to the maximization of the δ variables to **true**. We then note $\Phi_{\Rightarrow}$ the reduced formula for anti-alignments given by the following axioms:

$$\delta_{0,0,0} \quad \wedge \quad \bigwedge_{d>0} \neg\delta_{0,0,d} \tag{5.11}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \quad (\delta_{i+1,0,d+1} \Rightarrow \delta_{i,0,d}) \tag{5.12}$$

$$\bigwedge_d \bigwedge_{j=0}^{n} \quad (\delta_{0,j+1,d+1} \Rightarrow \delta_{0,j,d}) \tag{5.13}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} = v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d} \Rightarrow \delta_{i,j,d}) \tag{5.14}$$

$$\bigwedge_d \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} \quad [u_{i+1} \neq v_{j+1}] \Rightarrow (\delta_{i+1,j+1,d+1} \Rightarrow (\delta_{i+1,j,d} \wedge \delta_{i,j+1,d})) \tag{5.15}$$

Optimal solutions of this reduction give the exact edit distance between two sequences. We note $val(s)$ the distance returned by a solution s to present the following lemma.

Lemma 10. *The maximal value obtained by maximizing $val(s)$ over the solutions of $\Phi_{\Rightarrow}$ is equal to the maximal anti-alignment distance obtained using $\Phi_{\Leftrightarrow}$. Formally:*

$$\max_{s \in sol(\Phi_{\Rightarrow})} val(s) = \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s') \ . \tag{5.16}$$

Proof. Similarly to the reduction to $\Phi_{\Leftarrow}$ in Lemma 1, we show that $\Phi_{\Leftrightarrow}$ and $\Phi_{\Rightarrow}$ define the same anti-alignment distance when we maximize $val(s)$:

1. $\max_{s \in sol(\Phi_{\Rightarrow})} val(s) \geq \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: The proof is exactly the same as for multi-alignments (see Lemma 1).

2. $\max_{s \in sol(\Phi_{\Rightarrow})} val(s) \leq \max_{s' \in sol(\Phi_{\Leftrightarrow})} val(s')$: The idea of the proof is similar to $\Phi_{\Leftarrow}$ reduction. We create $s' \in sol(\Phi_{\Leftrightarrow})$ such as, for $s \in sol(\Phi_{\Rightarrow}), \quad val(s) \leq val(s')$. This is proved by induction with : $\forall_{i,j,i+j \leq n} \quad \forall_d \quad s(\delta^{\sigma}_{i,j,d}) \Rightarrow (dist(\langle u_1, \ldots, u_i \rangle, \langle \sigma_1, \ldots, \sigma_j \rangle) \geq d)$.

□

Prefix Heuristics and Impact on Precision Measure

Finally the heuristics on prefixes given in Section. 3.3.5 can also be applied for anti-alignments. Indeed, one can compute prefix anti-alignments only and knows that the beginning of its model, in term of runs, is precise or not.

Like for multi-alignment, solving the problem of anti-alignment on prefixes dramatically improves efficiency, and already gives very relevant results in practice.

Furthermore, the optimal prefix based anti-alignment brings a lower bound for model precision under the assumption that the model is a sound workflow net, i.e., every prefix can be completed to reach the final marking. The intuition is that the most misaligned full run that we can expect is the optimal anti-alignment found on the prefixes, completed

with a sequence of perfectly misaligned actions. Following this intuition, one can deduce a lower bound on the precision for the full model and full log. This lower bound is quite tedious to compute in the general case but we illustrate it in a restricted case. We explain it in a simplified case of a model N and a log L where all log traces and full runs have the same length $2n$.

Proof. Let γ' be an anti-alignment found for the prefixes of size n, i.e. γ' is a prefix of size n for N, maximizing the $dist(\gamma', L')$ where L' is the truncated log. Let $\sigma' \in L'$ achieving $dist(\gamma', L')$, such that $dist(\gamma', L') = dist(\gamma', \sigma')$. Consider now any full run γ of N. Let γ_1 and γ_2 be the prefix and suffix respectively of size n of γ, such that $\gamma = \gamma_1 \cdot \gamma_2$ and $|\gamma_1| = |\gamma_2| = n$. By assumption, the distance between γ_1 and the truncated log L' is less or equal to $dist(\gamma', L')$. This means that there exists a log trace $\sigma \in L$ whose prefix σ_1 of size n satisfies $edits(\gamma_1, \sigma_1) \leq edits(\gamma', \sigma')$. From this we deduce that

$$edits(\gamma, \sigma) \leq edits(\gamma', \sigma') + 2n \tag{5.17}$$

Indeed, one way to edit γ to σ is to delete all the suffix γ_2 (n deletions), then edit γ_1 to σ_1 ($\leq edits(\gamma', \sigma')$ actions), and finally complete σ_1 to σ (n insertions). It follows that

$$dist(\gamma, L) \leq \frac{edits(\gamma', \sigma') + 2n}{4n} = \frac{1}{2}\left(\frac{edits(\gamma', \sigma')}{2n} + 1\right) = \frac{1}{2}(dist(\gamma', \sigma') + 1) \tag{5.18}$$

and

$$P^{\epsilon}_{aa}(N, L) \geq 1 - \frac{dist(\gamma', \sigma') + 1}{2(1+\epsilon)^{2n}}\,. \tag{5.19}$$

□

Notice that we did not compute this lower bound in the experiment part because the general case (where traces and runs have various lengths) is tedious to obtain. In Section 3.3.5, we give the prefix-based anti-alignments and the precision of those prefixes only, i.e., we reduce the problem to the set of prefixes. Definition 33 is then given by

$$P^{/n}_{aa}(N, L, n) = 1 - \sup_{\gamma \in Prefixes(N,n)} dist(\gamma, Truncate(L, n)) \tag{5.20}$$

where $Prefixes(N, n)$ denotes the set of prefixes of length n for the process model N, and $Truncate(L, n)$ denotes the log L with all traces truncated to length n. This precision on prefixes can already help one to analyze the *beginning* of a process model. However this approaches is limited because the end of runs might play a large role in precision of some process models. In the next section, we present a second algorithm that proposes full run anti-alignment approximation.

5.3.2 An A* Algorithm based on the Discounted Edit Distance for Approximating Anti-Alignments

The second algorithm to get anti-alignments uses an A* algorithm and the discounted edit distance. The overall idea is similar to the method presented in Section. 3.4 for alignments and multi-alignments, however the impact on this computation is higher due to the complexity of finding anti-alignments.

With the large amount of different behaviors in logs, process models tend to be large and contain a lot of choice, concurrency and loop behaviors. Naive exploration of the runs of a model has to consider huge number of candidates for anti-alignments. We reduce this exploration by using the discounted edit distance which assigns higher cost to edits that appear at the beginning of the sequences.

We recall that for $\theta = 1$, the discounted edit distance is the Levenshtein distance. However, for larger values of θ, edits at the beginning of the sequences are more costly than edits at the end because of the exponent k based on the position of the edits.

By assigning higher cost to edits at the beginning of the sequences, the discounted parameter θ allows one to select efficiently prefixes of runs which may be continued to promising anti-alignments. Then revelant values for θ are slightly larger than 1 and close to 1 if the purpose is to approximate the Levenshtein edit distance.

Example 5.3.1 (Discounted Edit Distance for Anti-alignments)**.** *For instance, for $\theta = 2$, an edit at position k costs more than the sum of all next edits of position $k' > k$. The best anti-alignment for $L3$ and $N3$ of Fig. 5.1 is $\langle a, b, c\rangle$ which is strongly more deviant in the beginning of the run with activity a. Its minimal distance to the log is $\theta^{-0} = 1$ and cannot be topped to another run despite summing edits. However, for lower values of the discounted parameter θ, like $\theta = 1.10$, the distance finds modeled behavior $\langle b, e, d, \tau\rangle$ as the further one from the log $L3$, like when using the Levenshtein edit distance.*

To get anti-alignments of a process model N and a log L using the discounted distance, we present Alg.7 that plays the firing sequences of the process model in order to find a full run γ such that :

$$\sup_{\gamma \in Runs(N)} \min_{\sigma \in L} \mathcal{D}_{\theta,\epsilon}(\gamma, \sigma) \tag{5.21}$$

where $\mathcal{D}_{\theta,\epsilon}(\gamma, \sigma) = \frac{\mathcal{D}_{\theta}(\gamma,\sigma)}{(1+\epsilon)^{|\gamma|}}$ to penalize long runs in case of loops in N.

Technical Details

This penalization on long runs is only required to anti-alignments as alignments and multi-alignments will always converge to approach the log input.

The algorithm maintains a priority queue of prefixes of runs, implemented as a heap of tuples $\langle \gamma, m, d\rangle$, where γ is the prefix, m is the state that it reaches, and d is the priority

with which it should be treated. This priority d is defined as the quantity:

$$h_{\theta,\epsilon}(\gamma, L) \stackrel{\text{def}}{=} \min_{\sigma \in L} \left(\mathcal{D}_{\theta,\epsilon}(\gamma, \sigma) + \frac{\theta^{-|\gamma|-|\sigma|}}{\theta - 1} \right) \tag{5.22}$$

that bounds from Eq. 5.21 the value $\min_{\sigma \in L} \mathcal{D}_{\theta,\epsilon}(\gamma', \sigma)$ that any full run γ' having γ as prefix can achieve.

New prefixes are obtained by firing the transitions of the process model (line 14 of Alg.7) and the algorithm terminates when the queue is empty or no candidate prefix can improve the best value obtained so far (line 8).

Algorithm 7: Computation of Anti-Alignment by using the Discounted Distance

Input : $N = (P, T, F, \lambda, m_0, m_f)$: process model, L: log,
θ: discount parameter,
ϵ: long run limit parameter

```
1  Q ← {⟨⟨⟩, m_0, h(⟨⟩, L)⟩}        // Heap of open states ordered by distance,
     maximum is placed on top
2  B_γ ← undefined                          // Current best anti-alignment
3  B_δ ← −∞                          // Current best distance to reach m_f
4  while Q ≠ ∅                                // While not all states visited
5  do
6      ⟨γ, m, d⟩ ← Q.pop()                          // Next state maximizing d
7      if d ≤ B_δ then
8          break while                  // No state is going to improve B_δ
9      if m == m_f then
10         δ ← min_{σ∈L} D_{θ,ε}(γ, σ)                       // Exact distance δ
11         if B_δ < δ then
12             B_γ ← γ                          // New best anti-alignment
13             B_δ ← δ                                  // Update distance
14     for t ∈ T with m[t⟩m' do
15         γ' ← γ · t                                    // Get new prefix
16         d' ← h(γ', L)          // Get possible distance of γ' to L
17         Q ← Q.insert(⟨γ', m', d'⟩)                    // Place new state
```

Output: $\mathcal{B}_\gamma$: best anti-alignment,
$\mathcal{B}_\delta$: minimal distance of $\mathcal{B}_\gamma$ to L

Proof of optimality

Let us first prove the announced fact: for every full run γ' having γ as prefix, $h_{\theta,\epsilon}(\gamma, L) \geq \min_{\sigma \in L} \mathcal{D}_{\theta,\epsilon}(\gamma', \sigma)$. Let γ, $\gamma' = \gamma.u$ and σ, we show that $\mathcal{D}_{\theta,\epsilon}(\gamma, \sigma) + \frac{\theta^{-|\gamma|-|\sigma|}}{\theta-1} \geq \mathcal{D}_{\theta,\epsilon}(\gamma', \sigma)$.

This is because, in order to edit γ' to σ, one can edit its prefix γ to σ (at cost $\mathcal{D}_{\theta,\epsilon}(\gamma,\sigma)$) and then delete the letters of u one by one. The first deletion costs $\theta^{-|\gamma|-|\sigma|-1}$, the second one $\theta^{-|\gamma|-|\sigma|-2}$... and the sum of these costs is bounded by $\sum_{i=1}^{\infty}\theta^{-|\gamma|-|\sigma|-i} = \frac{\theta^{-|\gamma|-|\sigma|}}{\theta-1}$.

The algorithm satisfies the following invariant, which holds before and after each iteration of the while loop: either current best value is optimal or every optimal anti-alignment has a prefix in the queue Q. It is preserved at each iteration for the following reason. Let $\langle\gamma, m, d\rangle$ pop from Q and assume γ is the prefix of an optimal full run γ'. Either $\gamma' = \gamma$ (then $\min_{\sigma\in L}\mathcal{D}_{\theta,\epsilon}(\gamma,\sigma)$ is compared with $\mathcal{B}_\delta$) or γ' has a prefix of the form $\gamma.t$, which is queued. The while loop is broken (line 8) only when $d \leq \mathcal{B}_\delta$. As stated before, this implies that no γ' having γ as prefix will beat the current best value; and this also holds for all the other prefixes remaining in the queue since their value is smaller than d.

Termination: once $\mathcal{B}_\delta > 0$, any γ longer than $\frac{\log\frac{\theta}{(\theta-1)\mathcal{B}_\delta}}{\log(1+\epsilon)}$ popping from Q breaks the while loop. Indeed, this implies $(1+\epsilon)^{|\gamma|} \geq \frac{\theta}{(\theta-1)\mathcal{B}_\delta}$. Moreover, for every σ, $\mathcal{D}_\theta(\gamma,\sigma) \leq \sum_{i=0}^{\infty}\theta^{-i} = \frac{\theta}{\theta-1}$; hence $\mathcal{D}_{\theta,\epsilon}(\gamma,\sigma) \leq \mathcal{B}_\delta$ Since only finitely many prefixes are shorter, and no prefix is queued twice, the termination of the algorithm is guaranteed as soon as $\mathcal{B}_\delta > 0$, i.e. a full run $\gamma \notin L$ has been found.

Example 5.3.2 (Drawing Example of the A* Heuristic Function)**.** *This example illustrates the use of the fraction $\frac{\theta^{-|\gamma|-|\sigma|}}{\theta-1}$ in the definition of $h_{\theta,\epsilon}$. Let $\theta = 1.10$ and $\epsilon = 0$ for log $L3$ and model $N3$ of Fig.5.1. The open set Q contains, at some point, states $\langle\langle b\rangle, \{p_1, p_2\}, d_b\rangle$ and $\langle\langle a\rangle, \{p_3\}, d_a\rangle$ of concurrent prefixes $\langle b\rangle$ and $\langle a\rangle$.*
Let's try to consider only their distance to the log, i.e., $d_b = \min_{\sigma\in L}\mathcal{D}_{1.10,0}(\langle b\rangle,\sigma) = 0.83$ and $d_a = \min_{\sigma\in L}\mathcal{D}_{1.10,0}(\langle a\rangle,\sigma) = 2.73$. Prefix $\langle a\rangle$ is the best current anti-alignment and the algorithm continues with this prefix (line 6) until the final marking. The final run is $\mathcal{B}_\gamma = \langle a, b, c\rangle$ and $\mathcal{B}_\delta = 1$. Prefix $\langle b\rangle$ is then forgotten at line 7 because its current distance is lower than $\mathcal{B}_\delta$. However, for $\theta = 1.10$, the optimal anti-alignment is indeed $\langle b, e, d, \tau\rangle$. By adding the fraction $\frac{\theta^{-|\gamma|-|\sigma|}}{\theta-1}$ in d_b and d_a given in function h at line 16 which prevents best suffixes, $d_b = 7.51$ and prefix $\langle b\rangle$ is now handled at line 7 and becomes the best anti-alignment.

Heuristic to Further Reduce the Search Space

Like for multi-alignments, we use a parameter μ to set a maximal number of times we reach the same marking. Then this reduces loops and choice explorations.

Technical Details

The A*-based Algorithm allows one to get non-optimal anti-alignments for large model and thus allowing computing anti-alignment-based precision for real-life instance which was not possible before.

Obtaining Precision with Discounted Anti-alignments

The main purpose of finding anti-alignments is to compute precision of process models. The present algorithm finds fast anti-alignments not necessarily optimal. In return, the technique is able to work with real instances. Once an anti-alignment is found, one can use it to compute precision of its model by using the classical Levenshtein based precision as defined in Section 2.2.2. The expression

$$\sup_{\gamma \in Runs(N)} \min_{\sigma \in L} \Delta_\epsilon(\gamma, \sigma) \tag{5.23}$$

of Def.33 is approximated by

$$\min_{\sigma \in L} \Delta_\epsilon(\gamma, \sigma) \tag{5.24}$$

for the run γ computed by Alg.7. In practice, we observe perfect matching results for some instances.

5.4 Anti-alignment and Precision Experiments

Before studying anti-alignment use in clustering context, we give some experimentation of the algorithms to show their efficiency and accuracy. Moreover, we compare our precision measure to the state-of-the-art methods.

The SAT-based approach is given in both Darksider and da4py. We employed the latter which implements the formula reduction presented in Section 5.3.1. The A*-based algorithm is given in a branch of pm4py. The experiments have been done on a MacBook air 2017 model with a 1.8 GHz Intel ® Core^{TM} i5 CPU and 8G RAM.

5.4.1 Comparison of the Results Obtained with Different Settings

First, we present a set of experiments for anti-alignment and precision computation for the two algorithms with different settings. The aim of this section is to show the impact of the parameters such that one can consider it when using our methods. We use the artificial log $L_a = \{\langle A, B, D, E, I\rangle, \langle A, C, D, G, H, F, \rangle, \langle A, C, G, D, H, F\rangle, \langle A, C, H, D, F, I\rangle, \langle A, C, D, H, F, I\rangle\}$ which was already used in Section. 3.5.2 and its corresponding models. In this section, we consider all the models presented in [129, 8]. We give in Fig. 5.3 to 5.6 the missing ones.

Figure 5.3: Single.

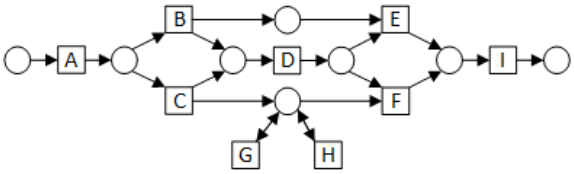


Figure 5.4: Model with G and H as self-loops.

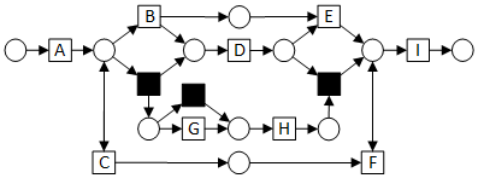


Figure 5.5: A model where C and F are in a Loop, but Need to Be Executed Equally Often to Reach the Final Marking.

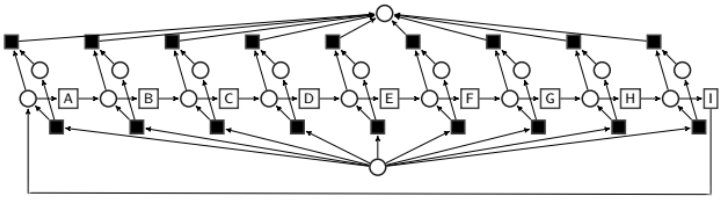


Figure 5.6: Round-robin.

Heuristics of the SAT Encoding

To deal with large logs, the SAT-based algorithm has a variant of anti-alignments that works on a fixed size of prefix.

Table 5.1 figures the impact of this heuristic for prefix anti-alignment of a size to 5, 10 and 15 and the corresponding adapted precision P'^{n}_{aa} given by Equation (5.20). With the use of prefix only, the method does not require full runs. This can be notified for models with loops for which anti-alignments do not reach the final marking. As the size of the run is fixed, parameter ϵ, used to limit the size of the run, is not required and corresponds to $\epsilon = 0$.

In most cases, results for distance to 10 and 15 are similar. However, we remark two types of differences. When using a small size of prefix, a model with a loop is qualified as more precise than when the size of prefix is large, thus alleviating the use of loops. For well precise model, reducing the size of the prefix can induce a decrease of the metric. For instance, for the generative model, Fig. 3.8, the transitions F and I are well presented in the traces and do not raise the distances, thus raising the denominator of the normalization of the precision.

Model		Size of Prefix	Anti-alignments	$P_{aa}^{/n}$	Time (s)
Fig. 3.8	Generating model	5	$\langle A, C, G, H, D\rangle$	0.800	1.58
		10	$\langle A, C, G, H, D, F, I\rangle$	0.923	12.73
		15	$\langle A, C, G, H, D, F, I\rangle$	0.923	37.89
Fig. 5.3	Single	5	$\langle A, B, D, E, I\rangle$	1.000	0.68
		10	$\langle A, B, D, E, I\rangle$	1.000	6.66
		15	$\langle A, B, D, E, I\rangle$	1.000	21.94
Fig. 3.10	Flower model	5	$\langle \tau, G, G, E, B\rangle$	0.300	1.45
		10	$\langle \tau, G, G, G, G, G, G, G, G, G\rangle$	0.176	12.59
		15	$\langle \tau, G, G, G, G, G, G, G, G, G, G, G, G, G, G\rangle$	0.136	44.88
Fig. 3.11	Separate traces	5	$\langle A, C, D, G, H, F, I\rangle$	1.000	2.76
		10	$\langle A, C, D, G, H, F, I\rangle$	1.000	26.56
		15	$\langle A, C, D, G, H, F, I\rangle$	1.000	97.83
Fig. 3.9	G,H in parallel	5	$\langle A, C, \tau, D, \tau\rangle$	0.800	1.07
		10	$\langle A, C, D, \tau, \tau, F, I\rangle$	0.923	8.83
		15	$\langle A, C, \tau, D, G, F, I\rangle$	0.929	40.58
Fig. 5.4	G,H as self-loops	5	$\langle A, C, G, G, G\rangle$	0.600	0.85
		10	$\langle A, C, G, G, G, G, G, G, G, G\rangle$	0.353	9.66
		15	$\langle A, C, G, G, G, G, G, G, G, G, G, G, G, G, G\rangle$	0.272	37.51
Fig. 3.12	D as self-loop	5	$\langle A, B, D, D, D\rangle$	0.600	0.93
		10	$\langle A, C, D, D, D, D, D, D, D, D\rangle$	0.375	11.17
		15	$\langle A, C, D, D, D, D, D, D, D, D, D, D, D, D, D\rangle$	0.286	48.68
Fig. 3.13	All parallel	5	$\langle \tau, G, C, E, B\rangle$	0.300	1.06
		10	$\langle \tau, D, I, F, B, H, A, G, C, E\rangle$	0.313	13.82
		15	$\langle \tau, F, H, G, C, I, E, D, B, A, \tau\rangle$	0.353	77.59
Fig. 5.5	C,F equal loop	5	$\langle A, C, \tau, D, \tau\rangle$	0.600	1.69
		10	$\langle A, C, B, D, E, F, I\rangle$	0.833	12.62
		15	$\langle A, C, B, D, E, F, I\rangle$	0.833	38.66
Fig. 5.6	Round-robin	5	$\langle \tau, D, E, F, G\rangle$	0.500	3.84
		10	$\langle \tau, E, F, G, H, I, A, B, C, D\rangle$	0.529	28.81
		15	$\langle \tau, B, C, D, E, F, G, H, I, A, B, C, D, E, F\rangle$	0.500	99.84

Table 5.1: Anti-alignment Prefixes of size 5, 10 and 15 for the Artificial Models of [129] and Log L_a

Finally, we invite readers to compare anti-alignment *prefix based* precision to exact anti-alignment precision (lines SAT/n and SAT in Table 5.6). We see that for most models, approximated precision tends to be close to optimal one for a great gain of runtime. Indeed, while using the optimal algorithm for precision (Alg. 5), many anti-alignments have to be computed until the metric converges, thus by increasing the size of the run.

Another heuristic of the SAT-based algorithm is the threshold on the number of edits introduced on page. 36. The idea is to reduce the maximal number of edits max_d of the Levenshtein distance in order to considerably reduce the size of the SAT formula.

We present in Table 5.2 a concrete example of the consequences of this parameter. For log L_a, model of Fig. 5.4 and a prefix of run to 10, the maximal number of edits is 17 (the maximal length in log is 7). The first row of Table 5.2 shows the exact anti-alignment which is at distance 11 to any log trace. When we reduce the number of edits, the tool might still find the same anti-alignment. Indeed for next row of the table, max_d to 5, the tool finds anti-alignment $\langle A, C, G, G, G, G, G, G, G, G\rangle$ which is indeed *at least* at distance 5 to any log trace. In fact, this anti-alignment is optimal and it is at distance 11, but has

Parameter max_d, Threshold on the Number of Edits	Returned Anti-alignment	Returned Threshold Lower Bound Distance to Log	Correct Minimal Distance to Log	Precision	Time (s)
17	$\langle A, C, G, G, G, G, G, G, G, G \rangle$	11	11	0.353	8.95
5	$\langle A, C, G, G, G, G, G, G, G, G \rangle$	5	11	0.353	2.65
5	$\langle A, C, D, G, G, G, G, G, H, H \rangle$	5	7	0.588	2.48
5	$\langle A, C, D, H, G, H, G, G, G, F \rangle$	5	5	0.706	2.60

Table 5.2: Consequences of the Threshold on Number of Edits of Leventshein Distance for log L_a of Fig. 3.7, model of Fig. 5.4 and a prefix of run to 10. The last line is forced to reach the final marking.

been returned by chance. In fact, for max_d to 5, the tool also returned the two last lines of the table where the optimal anti-alignment has not been found. For them, the correct distance to the log is not 11 which implies a difference in precision.

Experiment Conclusion

We motivate users to use the prefix based method and a maximal number of edits which improves a lot the runtime. A preliminary study of the length of full runs of the models can help one to set the size of the prefix.

The Discounted Parameter and Heuristic of the A*-based Approach

The A*-based algorithm has three parameters : the discounted parameter θ which, for high values, brings to a prefix search reduction, parameter ϵ that helps to deal with loops in models and the threshold μ which limits the number of times the algorithm can reach the same marking.

We experimented different values of these parameters for the artificial log L_a and two of its associated models specifically chosen for showing the impact of the settings: the generating model and the flower model. The generative model has a finite set of runs obtained by its several choice structures. In opposition, the flower model represents an infinite language of its transition labels which are all connected to the same place. We choose those models to present a normal case versus a complex and imprecise model.

In Tab.5.3, we show how changing the discounted parameter θ can improve the runtime. For large θ, prefixes cost more than suffixes, thus allowing the algorithm to considerably reduce the exploration. This aspect appears very clearly for the flower model.

The anti-alignments found for the flower model are not long because of the parameter ϵ set to 0.01. Then we observe that the found precision is quite high for this model which is very imprecise. By setting ϵ to larger values, we would get longer anti-alignments, thus providing more significant value of precision. However, due to the possible combinations of longer runs, the algorithm would blow up.

Model	θ	Anti-alignment	P^{ϵ}_{aa}	Time (s)
Generating	1.1	$\langle A, C, G, H, D, F, I \rangle$	0.928	0.02
model	1.5	$\langle A, C, G, H, D, F, I \rangle$	0.928	0.01
	2.0	$\langle A, C, G, H, D, F, I \rangle$	0.928	0.01
Flower	1.5	$\langle \tau, F, F, F, E, \tau \rangle$	0.400	29.85
model	2.0	$\langle \tau, F, F, F, F, \tau \rangle$	0.372	4.67

Table 5.3: Computation for Different Values of the Discounted Parameter θ and $\epsilon = 0.01$

Model	μ	ϵ	Anti-alignment	P^{ϵ}_{aa}	Time (s)
Flower	100	0.01	$\langle \tau, F, F, E, H, \tau \rangle$	0.400	2.50
model	100	0.001	$\langle \tau, F, F, F, F, F, F, A, A, H, \tau \rangle$	0.302	4.61
	5	0.01	$\langle \tau, F, F, F, F, \tau \rangle$	0.372	0.07

Table 5.4: Reducing the Search Space with Parameter μ for Different ϵ Values and $\theta = 1.5$

Tab.5.4 aims at showing the benefit of the limit μ on the number of exploration of prefixes reaching the same marking. We see that for $\theta = 1.5$ and $\epsilon = 0.01$, the runtime of computing an anti-alignment for the flower model is divided by 10 when setting μ to 100. The runtime improvement by using μ is prominent and allows to explore longer runs (for instance Tab.5.4 shows a result where $\epsilon = 0.001$). Finally, the last line of Tab.5.4 presents an experiment where μ is very small and still provides a relevant anti-alignment (just a bit shorter).

Experiment Conclusion

Increasing the value of the discounted parameter θ prioritizes the misalignment of prefixes, reducing the search space for a gain in runtime and a loss on the quality of the anti-alignment, where optimal anti-alignments are based on the Levenshtein distance. To further improve runtime and reduce the reachibility exploration, parameter μ limits the number of times every marking is reached. We obverse that very low values of μ give good outputs. Finally, we advise setting ϵ to equivalent value of the ϵ used in precision.

Side-by-side Comparison

Now that we have presented the approximation impacts, we take advantage of them and compare the results to the optimal results for all the artificial models associated to the log L_a.

In Tab.5.5, column "Method" precises if anti-alignments have been computed with the optimal algorithm of the SAT approach (Alg. 5 where we stop the exploration when m of line 3 stays stable for 10 runs in a row), its approximation version that computes prefixes of anti-alignments only noted with $/n$ (Equation (5.20)), or the A*-based approach that uses the discounted distance as heuristic function (Alg. 7). We set the prefix size of the

SAT encoding to 11 and require the algorithm to reach the final marking such that we get full run anti-alignments. About the A* algorithm, we set the parameter θ to 1.5, ϵ to 0.01 and μ to 10. Lines with $-$ marks were not computable on the selected machine for this experiments due to memory space.

Observations are significant: the A* algorithm runs much faster and obtains, in most times, the optimal results. Large runtime of the SAT lines are explained by the fact that Alg. 5 iteratively computes anti-alignments by increasing the size of the run until the precision measure converges. Thanks to the fixed size of the run, we observe that the prefix heuristic of the SAT encoding noted SAT/n gets very good results. This result is obtained because we set a size of run that allows to reach the final marking by exploring enough firing sequences of the models. However this information is usually unknown.

Model	Method	Anti-alignment	P_{aa}	Time (s)
Generating model	SAT	$\langle A, C, G, H, D, F, I\rangle$	0.928	266.09
	SAT/n	$\langle A, C, G, H, D, F, I\rangle$	0.923	16.51
	A*	$\langle A, C, G, H, D, F, I\rangle$	0.928	0.01
Single	SAT	$\langle A, B, D, E, I\rangle$	1.000	71.39
	SAT/n	$\langle A, B, D, E, I\rangle$	1.000	9.92
	A*	$\langle A, B, D, E, I\rangle$	1.000	0.01
Flower model	SAT	$\langle \tau, G^{23}, \tau\rangle$	0.295	2244.97
	SAT/n	$\langle \tau, G, G, G, G, G, G, G, G, G, \tau\rangle$	0.222	19.89
	A*	$\langle \tau, F, F, F, E, \tau\rangle$	0.400	0.03
Separate traces	SAT	$\langle A, C, G, D, H, F, I\rangle$	1.000	288.137
	SAT/n	$\langle A, C, G, D, H, F, I\rangle$	1.000	42.26
	A*	$\langle A, B, D, E, I\rangle$	1.000	0.01
G,H in parallel	SAT	$\langle A, C, G, H, D, F, I\rangle$	0.928	290.95
	SAT/n	$\langle A, C, D, G, \tau, F, I\rangle$	0.923	20.00
	A*	$\langle A, C, G, H, D, F, I\rangle$	0.928	0.04
G,H as self-loops	SAT	$-$	0.496	$-$
	SAT/n	$\langle A, C, G, D, G, G, G, G, G, F, I\rangle$	0.667	14.95
	A*	$\langle A, C, G^{9}, H, D, F, I\rangle$	0.631	0.15
D as self-loops	SAT	$-$	0.496	$-$
	SAT/n	$\langle A, B, D^{7}, E, I\rangle$	0.625	19.99
	A*	$\langle A, B, D^{10}, E, I\rangle$	0.588	0.17
All parallel	SAT	$\langle \tau, I, G, F, E, H, D, B, C, A, \tau\rangle$	0.420	2006.66
	SAT/n	$\langle \tau, I, E, F, D, H, C, B, A, G, \tau\rangle$	0.353	24.03
	A*	$\langle \tau, I, F, E, H, C, A, G, B, D, \tau\rangle$	0.525	4.08
C,F equal loop	SAT	$\langle A, C, B, D, E, F, I\rangle$	0.845	292.81
	SAT/n	$\langle A, C, B, D, E, F, I\rangle$	0.833	18.01
	A*	$\langle A, C^{7}, B, D, E, F^{7}, I\rangle$	0.502	0.67
Round-robin	SAT	$\langle \tau, D, E, F, G, H, I, A, B, C, \tau\rangle$	0.300	178.20
	SAT/n	$\langle \tau, E, F, G, H, I, A, B, C, D, \tau\rangle$	0.444	45.01
	A*	$\langle \tau, E, F, G, H, I, A, B, C, D, \tau\rangle$	0.502	0.01

Table 5.5: Comparison of Anti-alignments and Precision on Artificial Log L_a and its Associated Models. The A*-based Algorithm defined on the discounted distance is set with $\theta = 1.5$, $\epsilon = 0.01$ and $\mu = 10$. Optimal Precision for $\epsilon = 0.01$ (lines SAT) is given by Alg. 5 where we stop the search after 10 equal results in a row. SAT/n runs the SAT heuristic with a size of run to 11 and optimal number of edits for this size.

Differences between the found anti-alignments can be explained by the parameters. For instance, by using the flower model, the A* algorithm returns a shorter run due to parameter μ which strictly reduces the number of times the algorithm can reach a marking.

The big difference observed for the model entitled *C,F equal loop*, for which our algorithm finds a much worse anti-alignment, is due to the fact that this model is not safe, i.e., transitions labeled with C and F output a token in their input place, thus allowing running the transitions again. But, the SAT implementation does not consider unsafe Petri nets (it restricts their behavior to runs visiting only safe markings), and hence, cannot guarantee optimality for this model. The precision obtained with the discounted anti-alignment is then more accurate than the SAT-based version.

We note that the SAT/n sometimes returns lower value of precision than the optimal one SAT which is due to the normalization that works with the size of the run only for this prefix-based version while the optimal version incorporates the parameter ϵ in precision measure.

Experiment Conclusion

The A* algorithm that uses the discounted distance gives very fast anti-alignments which are close the optimal for measuring precision of models. The main differences appear when models have looped because the parameter μ of the A*-algorithm stops the exploration search when same markings are reached many times. Then, we advise to consider this parameter for getting precise precision measures.

5.4.2 Anti-alignment based Precision Compared to the State-of-the-art Methods

One motivation for computing anti-alignments is to measure precision of process models which is much discussed in the process mining community. In this section, we compare our precision measure and runtime, for optimum and approximation, with the state-of-the-art methods.

Optimum Precision Comparison

In Table 5.6, we give our optimal precision for the artificial inputs, already shown in the previous section, next to the state-of-the-art methods. We take the opportunity to present results for another value of epsilon, i.e, $\epsilon = 0.05$ and the Hamming-based anti-alignment precision previously developed [31]

Column P_a gives the precision values as defined in [4]. The P_{ET} and P_{ETC} metrics are from work of [5]. P_{ne} denotes the precision metric of [134]. The EMP measure is of [70]. The values MAP^3 and MAP^7 are defined in [8][2]. All metrics are presented in the related

[2]Notice that the metrics MAP^3 and MAP^7 are not applicable for one of the benchmarks of this paper,

Model		P_{ET}	P_{ETC}	P_a	P_{ne}	EMP	MAP^3	MAP^7	$P_{aa}^{H,.05}$	$P_{aa}^{L,.05}$	$P_{aa}^{L,.01}$
Fig. 3.8	Generating model	0.992	0.994	0.982	0.995	0.902	0.880	0.852	0.797	0.945	0.928
Fig. 5.3	Single trace	1.000	1.000	1.000	0.893	1.000	1.000	1.000	1.000	1.000	1.000
Fig. 3.10	Flower model	0.136	0.119	0.142	0.117	0.000	0.003	0.000	0.408	0.352	0.295
Fig. 3.11	Separate traces	1.000	0.359	1.000	0.985	1.000	1.000	1.000	1.000	1.000	1.000
Fig. 3.9	G,H in parallel	0.894	0.936	0.947	0.950	0.785	0.564	0.535	0.797	0.945	0.928
Fig. 5.4	G,H as self-loops	0.884	0.889	0.947	0.874	0.568	0.185	0.006	0.570	0.780	0.496
Fig. 3.12	D as self-loop	0.763	0.760	0.797	0.720	0.694	0.349	0.069	0.506	0.759	0.469
Fig. 3.13	All parallel	0.273	0.170	0.336	0.158	0.000	0.006	0.000	0.468	0.468	0.420
Fig. 5.5	C,F equal loop	0.820	0.589	0.839	0.600	–	–	–	0.751	0.881	0.845
Fig. 5.6	Round-robin	0.579	0.185	0.889	0.194	0.000	0.496	0.274	0.456	0.352	0.300

Table 5.6: Precision Measures of Artificial Models

work section.

Finally column $P_{aa}^{H,\epsilon}$ and $P_{aa}^{L,\epsilon}$ represent the anti-alignment based precision for the Hamming and Levenshtein distance computed with the SAT encoding and the optimal algorithm given in [30].

Clearly, the existing precision metrics do not agree on all models and do not always agree with the intuition for precision. While all methods rank the All-parallel model just after the Flower model, our AA-based precisions position the Round-Robin model in second place instead. This is due to the fact that our method found the anti-alignment $\langle \tau, \tau \rangle$ which is far from the log traces. In fact, this run does not exist in the first version of this model in [129] for which the situation would not occur.

Figure 3.11, which consists of all the traces in separate paths, is considered to have a low precision by the P_{ETC} metric while all the other methods describe this model as perfectly precise.

We observe that our methods give higher precisions than the rest for the flower model. This is due to the requirement of the τ transitions in order to get a full run of the model, thus alleviating the distance to the traces. A similar situation, with the same antidote, happens with the parallel model.

More generally, we note that our method, due to the parameter ϵ, gives a good balance for models with loops. The precision of those models are very mixed. We see on one side P_{ET}, P_{ETC}, P_a and P_{ne} are from 0.947 to 0.720 and on the other side, MAP gives a precision close to zero. However, all the methods agree on the ranking of the loops compared to the parallel model.

Now, notice that model 3.8 and 3.9 are considered to be equally precise with the anti-alignment precision, which is not the case for the other metrics. Model 3.9 contains the behaviors of model 3.8 plus some permutation and skip activities between G and H, thus adding runs in the models. The fact that our method finds the same precision is a consequence of the input log. Model precision is defined in accordance to the log. In this case, there is no anti-alignment of Fig. 3.9 that is further to the log traces than the anti-alignment of Fig. 3.8, despite their are more anti-alignments.

due to the existence of unbounded constructs.

Finally, we would like to point out the practical aspect of the parameter ϵ. See precisions of Figure 5.4 for $\epsilon = 0.05$ and $\epsilon = 0.01$. The score considerably drops from $P_{aa}^{L,0.05} = 0.780$ to $P_{aa}^{L,0.01} = 0.496$. For large ϵ values, traces with a thousand G's is possible in the model and our anti-alignment can penalize it a lot by allowing long runs.

Experiment Conclusion

The anti-alignment-based precision shows some weaknesses due to the parameter ϵ that helps to deal with loops in models but reduces theoretical accuracy of the measure as loops allow infinite language. However, this aspect might be well accepted in practice because loops in model are sometimes required.

Precision Approximation for Real-life Models

In this section, we present, for the first time, anti-alignment based precision on real-life models whose description can be found in Tab. 3.3. We recall that the SAT implementation of anti-alignment cannot deal with entire large logs due to the complexity of the encoding which confronts memory issues.

We show both computation time and precision measure in Tab. 5.7 and compare our work to the ETC [6], the MAP^3 [8] and the EMP [70] measures.

Log	Model	Precision				Runtime (s)			
		ETC	MAP^3	EMP	P_{aa}^{ϵ}	ETC	MAP^3	EMP	P_{aa}^{ϵ}
BPI2012	IM	0.561	0.492	0.602	0.761	513.28	6.94	44.84	79.73
	SM	0.915	0.196	0.538	0.753	438.65	6.98	27.50	176.69
BPI2019	IM	–	1.000	0.468	0.934	–	42.26	421.42	727.86
	SM	–	0.780	0.903	0.950	–	33.11	331.84	219.74
$BPI2020_{dd}$	IM	0.636	0.472	0.804	0.868	0.83	3.63	7.03	0.38
	SM	0.953	0.040	0.861	0.894	0.76	4.17	3.90	1.04
$BPI2020_{rp}$	IM	0.346	0.074	0.319	–	1.64	3.58	13.88	–
	SM	0.815	0.017	0.780	0.604	0.64	4.39	9.22	0.59

Table 5.7: Real-life Logs and Models Precision where the Anti-alignment Precision P_{aa}^{ϵ} is found with Discounting Anti-alignments and $\theta = 2$, $\epsilon = 0.01$, $\mu = 5$

We remark than any precision measure agrees for all the inputs which strengthens the interest of research about this metric in process mining. The MAP^3 measure finds a perfectly precise model, i.e., the IM model of BPI2019 log. Our method has a similar precision, 0.934, but found an anti-alignment for this model which can figure the language difference between the log and model. The EMP measure is stricter with a precision of 0.468. For this log, ETC precision of [7] lasts more than several hours, so we stopped the process. This method uses alignments whose complexity is related to the log complexity. As we can see in Tab.3.3, the BPI2019 log is the more complex log with 251 734 cases and 42 different type of activities.

Our method also has to deal with log complexity. More than 75% of our runtime presented in Tab.5.7 is used to compute the discounted distances. For the IM model of BPI2020$_{rp}$, our method also takes several hours. This model contains two parallel structures of 10 and 2 transitions that explode the space of search. But in several cases, our method even gives the fastest runtime.

Experiment Conclusion

Besides runtimes, it is not obvious to conclude which precision metric is more valuable. However, as a conclusion of this section, we want to point out that, in order to understand the precision of process models, a human requires more than a numerical value. We believe that anti-alignment is the added value of our approach because it provides explanation for the imprecision of models.

5.5 Impact of Precise Process Models on Clustering Results

In the previous chapter, we presented our clustering algorithms which are the main contribution and the focus of this thesis. As our approach relies on process models, the quality of these models are in the scope of our approaches. In this section we provide a set of experiments to draw the relation between conformance of process models and clustering outputs.

The clustering method presented in this thesis works with alignments. A first intuition is that good fitting models provide good clustering.

To check these hypotheses, we launched two experiments. First, we give in Tab. 5.8 a study of the model-based clustering results of three logs for each three different process models. The models have been learned with different algorithms or preprocessing methods. We note with *Prototypes* the models that have used the process model discovery framework presented in [57] where only some median traces are used to learn a model. Lines noted with *Filtering* show process models given by [8] which are learned with the preprocessing method of [35]. We computed the anti-alignment based precision and the alignment based fitness implemented in pm4py.

The clustering algorithms require a set of parameters. We use the size of an alignment to set the size of the run and the number of transition per cluster which is a generic way to ensure that we will find a full run. We set the maximal distance between the traces and the centroids to 2. The clustering algorithm stops when no cluster is found after 5 iterations in a row. We give the script in Appendix. B.3.

The results, which are given in Tab. 5.8, are not clear for BPI2012. However, we note that for BPI2013$_{cp}$ the two most fitting process models give the least non-clustered traces. We also notice that the most fitting model for this log is also the least precise. For BPI2020$_{dd}$, model P+HM has a lower precision and a lower fitness than model P+IM.

Log	Model Information	Precision	Fitness	Nomber of Clusters	nc
BPI2012	Prototypes + Inductive Miner (P+IM)	0.76	0.78	2	8443
	Filtering + Inductive Miner (F+IM)	–	0.98	1	9658
	Prototypes + Split Miner (P+SM)	0.75	0.78	1	9658
$BPI2013_{cp}$	Filtering + Heuristic Miner (F+HM)	0.91	0.94	2	80
	Filtering + Split Miner (F+SM)	0.88	0.99	1	96
	Filtering + Inductive Miner (F+IM)	0.91	0.82	2	105
$BPI2020_{dd}$	Prototypes + Inductive Miner (P+IM)	0.87	0.96	6	117
	Prototypes + Split Miner (P+SM)	0.89	0.86	5	350
	Prototypes + Heurisitc Miner (P+HM)	0.30	0.91	0	10500

Table 5.8: Comparison of Clustering Results along with Precision and Fitness Metrics for Different Real-life Inputs. Precision is the approximated anti-alignment based precision with $\theta = 1.1, \mu = 20$ and $\epsilon = 0.01$. The size of the run and the number of transitions per clusters is set with the size of an alignment. The distance between the traces and the centroids is 2.

Then this model is less conform in general which can explain the clustering result which is null. Finally, we observe that the correlation of the metrics for P+IM and P+SM follows the hypothesis of the trade-off between fitness and precision, for these models, they are inversely correlated. The most fitting model again allows clustering more traces.

Nonetheless, the conclusion of these experiments about the clustering outputs, like the number of clusters, is not clear because the addressed models are learned with different methods. This questions if the conformance of the models is the clue or the discovery method makes the clustering outputs varying. To answer this question, we decided to go further in the exploration and compare comparable models, i.e., models discovered with the same method but different parameters. Furthermore, we give these experiments for two different algorithms with no preprocessing methods to compare the discovery methods.

Tab. 5.9 shows the clustering of the 1000 first log traces of $BPI2013_{cp}$. The use of a sublog in this study only aims at reducing the runtime of the experiment process and do not have impact on the interpretation of the results. We launched two algorithms implemented in pm4py that come with a threshold parameter. For the inductive miner, one can select the *noise threshold* to add in the log. The heuristic miner has a *dependency threshold* which allows more or less relations in the learning stage.

First, we observe that adding noise with inductive miner discovery algorithm implies a increase in precision. The fitness of the models for this algorithm stays the same for different values of the threshold. As the fitness is low, we obtain bad clustering outputs.

The results of the heuristic miner are more interesting. It clearly shows that when precision decreases fitness raises. The number of non-clustered traces also fills our expectation: the more precise is the model, the least there are non-clustered traces.

Now the number of clusters varies without correlation with the other information. This aspect is due to the fact that the clusters are found with a sampling method. Some large centroids can be found and cluster many traces while small and precise centroids can only

Algorithm	Threshold	Precision	Fitness	Nomber of Clusters	nc
Inductive Miner	0.1	0.91	0.46	0	1000
	0.2	0.91	0.46	1	984
	0.5	0.91	0.46	0	1000
	0.8	0.93	0.46	0	1000
	0.9	0.93	0.46	0	1000
Heuristic Miner	0.9000	0.96	0.62	3	80
	0.9900	0.96	0.70	2	43
	0.9990	0.57	0.98	5	9
	0.9999	0.54	0.98	3	10

Table 5.9: Comparison of Clustering Results of the 1000 First Traces of Log BPI2020_{cp} for Different Process Models Learned with the Same Algorithms but Different Settings. Column "Threshold" gives the *noise threshold* for the inductive miner and the *dependency threshold* for the heuristic miner. Precision has been computed with the A* approximation method with $\theta = 1.1, \mu = 20$ and $\epsilon = 0.01$. The size of the run and the number of transitions per clusters is set with the size of an alignment. The distance between the traces and the centroids is 2.

cluster a limited number of traces thus allowing to cluster the left traces in another group. These clusters are found from the selection of the traces randomly given by the sampling at each iteration (see Fig. 4.7 for a quick recall of the algorithm).

Experiment Conclusion

The conclusion of these experiments is not obvious. Nevertheless, we note that too poorly fitting process model does not allow getting model-based clustering.

5.6 Opening: An Agile Framework for Model Repair with Anti-alignments

We close this thesis by presenting a framework that exploits anti-alignment for model repair in process mining by using the principles of agile method. We present this section as an opening because no repairing actions will be presented in this section but anti-alignments give the keys for future work in this direction.

5.6.1 Model Repair

Introduced in [56], *Model Repair* sits in between model discovery and conformance checking [28]. While model discovery brings an entire artefact to reflect the behaviors contained in log, it does not consider an existing process model as a baseline. In opposite, confor-

mance checking gives a model-to-log analysis and a process model is required as input. However no modification is made in the given model, even if the analysis depicts very bad modelization, because it is not the focus of conformance checking. Model repair stands for this purpose with the aim of finding a novel process model N', close to a reference model N, such that the conformance to the associated log L is improved from N to N'.

The main objective of searching for a model close to the existing one is to keep the knowledge of the process owner. Once a model is known and trusted by a company, many changes, as it appears when a novel model is discovering from scratch, involve further studies in the understanding of the novel result. In opposite, when only light changes are made in the reference model, the owner is quickly able to understand the novel artefact. Comparison between the two models can also highlight business indicators.

The task of repairing a process model can appear in various situations. Organizations change over time and create or remove their behaviors. Moreover, a process study can start with a hand-written process model which can then be improved with a study of the log. The Business Challenge of year 2015 tackled this situation where five cities followed a given process and resulted in different logs [127]. Then, the optimal repaired model of every cities is different but comparable to the original one. In any of the previous situations, we note that model repair can be seen as a tuning phase where the initial process model shows some weaknesses to represent the recorded behaviors.

Furthermore, model repair is a step that involves both the business and the scientific sides of process analysis. In a way the model should be improved to better reflect the recorded log but in another way we want to ensure that we do not decrease quality of the existing model because of errors in log. Thus, decisions of the business team are welcomed in the process of model repair. Study of [28] presents an incremental and interactive repair where the changes in the model are chosen by the user from a list of possibilities. We stand close to this work and propose an agile method for repairing a model thanks to the discovered anti-alignment of process models.

5.6.2 Agile Method

Incremental and interactive production emerges in the middle of the 19th century with the works of Walter Shewhart and William Edwards Deming [107, 48]. The foundation of *agile method* came later, in 2001, when 17 researchers put the ground in a manifesto to define the fundamental principles [60]. The agile method puts forward the interactions between the individuals involved in the project with the aim to give a working software at any stage. We list the fundamental principles of agile method as it is given in the manifesto [60] :

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

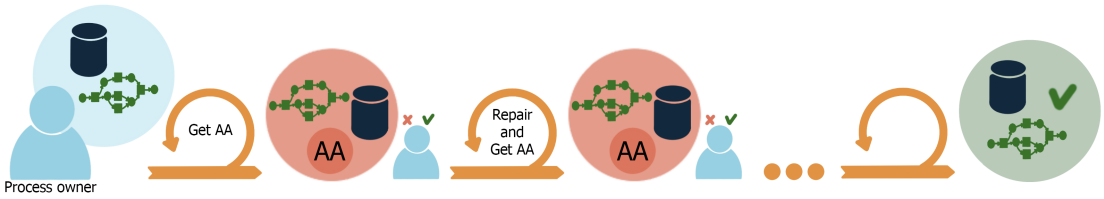


Figure 5.7: Agile Framework for Repairing Process Model with Anti-alignments

- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *Business people and developers work together daily throughout the project.*
- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress. Agile processes promote sustainable development.*
- *The sponsors, developers and users should be able to maintain a constant pace indefinitely.*
- *Continuous attention to technical excellence and good design enhances agility.*
- *Simplicity - the art of maximizing the amount of work not done- is essential.*
- *The best architectures, requirements and designs emerge from self-organizing teams.*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Those criteria have shown a great improvement in project development [84]. In process mining, some works have tackled agile methods but in the perspective of designing the agile method it-self [105]. Here, we want to use the agile method in the development of repairing the process model.

5.6.3 Anti-alignment, a Key for Repairing Process Model

We present in Fig. 5.7 the architecture of our proposal for repairing a process model with the use of anti-alignments. As anti-alignments are runs of the model that deviate from the recorded log, getting this modeled behaviors is a key for understanding both the real process and the modelization.

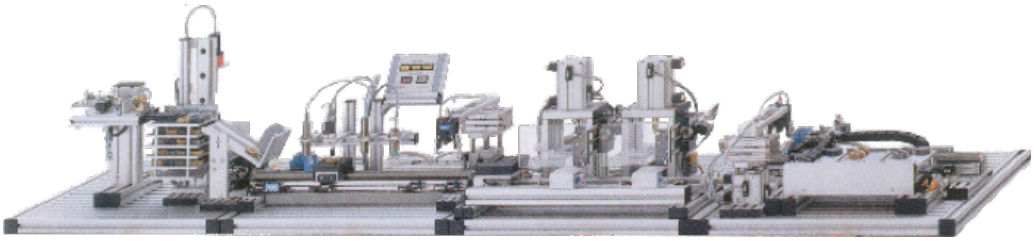


Figure 5.8: Bosh Engine

The process owner presents the log and the current model that may require to be repaired. By finding anti-alignments, we enable to check if the extra behavior of the process model is a misbehavior or not with respect to the real process. This knowledge is given by the process owner himself. This is where a discussion between the process mining expert and the process owner about the discovered anti-alignments of the current process model comes in. Thus, repairing steps can incorporate the knowledge to modify the model. This process presented in Fig. 5.7 loops until all the anti-alignments are either removed either accepted by the process owner. However, other terminations are also possible. For instance, one can use a threshold on the anti-alignment based precision to stop repairing the model, in accordance to the process owner agreement.

5.6.4 Case study: Anti-alignment Study of a Reactive Discrete Event System

Despite that we did not implement repairing actions, we have played the first step of the agile framework with another research team and discuss the value of anti-alignments. In this section, we develop our approach in the aim to present a real application of the use of anti-alignments.

The work entitled *Process Mining and System Identification in Automatic Control* is the result of a collaboration between Gregory Faraut from LURPA (Laboratoire Universitaire de Recherche en Production Automatisée) and our team from LSV (Laboratoire Spécification et Vérification). LURPA is a laboratory specialized in Automatic Production that designs and tests various engines.

In this work, we studied the behaviors of a reactive discrete event system given by the Bosch engine shown in Fig. 5.8. The machine aims at sorting out wheels of different materials. Any step of the process is recorded thanks to captors positioned on the machine.

One objective of LURPA research team is to find a precise model that capture all the behaviors of the machine and check the conformance of it in order to prove the accuracy of the discovered model. We focus on a station of the machine that especially has a lot of concurrent activities. A zoom on the station is given in Fig. 5.9. The station detects if the wheel has a plastic dowel and removes it if so. Otherwise, a dowel is added in the wheel. The station contains many steps. For instance, a check is done to verify that no wheel is already positioned in the dowel change area.

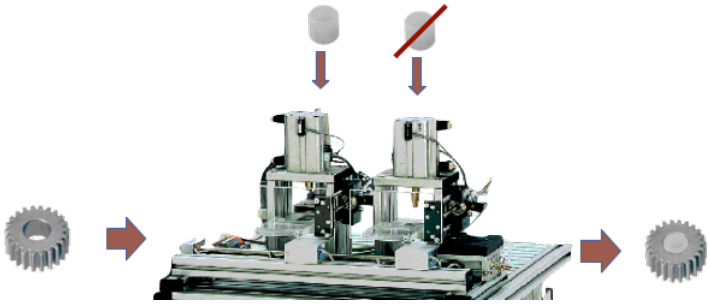


Figure 5.9: Third Station of the Bosch Engine

The log of this machine is recorded in term of captor changes given in bits. A change of a bit indicates the activation of an event in the machine. We extracted the sequences of events and generated a process model thanks to Apromore software [71].

Once that we have obtained a process model of the system, we launched the search of anti-alignments with respect to the event log. The expert of the Bosch engine appreciates the conformance checking artefacts that quickly help him to note the weaknesses of the discovered process model. Moreover, some anti-alignments were in fact possible by the system but did not appear in the log (yet). Then, this knowledge should be taken into account when repairing the process model.

Experiment Conclusion

This real-life example shows the interest of anti-alignments in the study of process model deviations. The knowledge of possible and non possible extra behaviors given by the process owner is a key interest for model repair.

5.7 Conclusion

In this chapter, anti-alignment was presented and compared to multi-alignment. While multi-alignment is a modeled sequence that is as close as possible to the log traces, anti-alignment plays the opposite role. The search of one of the most deviant run of a process model with respect to a log involves to check all the modeled behaviors which is an infinite set when process models contain loops. To deal with this limitation, we propose a penalization of very long runs.

The conformance checking artefacts are related and shared the two presented algorithms with slight differences. First, a MinSAT-based encoding allows moving from multi-alignments to anti-alignments with a change in the minimization problem. The negative of the optimization problem for multi-alignment gives anti-alignments. Second, an A*-algorithm explores the process model and prioritizes runs that optimize a specific distance to the log traces. For anti-alignment, the runs that maximize the minimal distance to

the log traces are prioritized. This latter algorithm allows computing approximated anti-alignment for real-life inputs for the first time.

Apart from algorithms, anti-alignment is a key conformance checking artefact that helps one to measure precision of process models. In this chapter, we show several experiments of this metric and compared it to the state-of-the-art methods. However the experiments that relate clustering and precision of process models did not lead to conclusive results.

Finally, we presented an agile framework for model repair with the use of anti-alignments. We experimented the first step and the conclusion was promising because the expert approved the approach and wanted to see repaired models that disable the deviant runs that we have identified. This encourages us to go in this direction for future work.

Chapter 6

Conclusion

Chapter Overview

This chapter summarizes the contributions of the thesis and lists the open issues and future works.

6.1 General Conclusion of the Contributions

This thesis gives a study of conformance checking artefacts in the context of model-based clustering of log traces. The main theme is the relationships between process models and event data given by the artefacts. We explored *alignment*, *multi-alignment* and *anti-alignment* to extract good representatives of process instances that we call *model-based trace variants*. The thesis provides formal definitions of our approaches associated to optimal algorithms which allows presenting proofs of concept of the methods. The algorithms are encoded as SAT instances which contributes to a recent family of algorithmic methods for conformance checking in the line of [31, 19]. In addition, heuristics of these algorithms and A*-based approximating algorithms are developed. They enable to work on large real-life inputs for a small lost of quality. All methods have been experimented and compared to several state-of-the-art approaches for artificial and real-life logs.

In summary, the following conformance checking artefacts have been developed:

Multi-alignment: presented in Chapter 3, multi-alignment artefact is a unique modeled sequence that represents a set of log traces. This modeled sequence is extracted from an input process model of the corresponding log. Its maximal distance to any log trace is minimized and the found artefact gives a median modeled view of the log. Multi-alignment showed relevant results when the log is homogeneous. However, for a varied set of traces, the artefact fails to represent the recorded behaviors because it gives a median modeling whom distance is far from all the traces.

Alignment: alignment artefact is not a novelty of conformance checking. In this thesis, we used alignment as a baseline of our clustering methods. The artefact allows to relate the traces to the model-based centroids. When a log contains different behaviors, the presented clustering methods enable to separate them in clusters based on the given process model. Each cluster is associated to a modeled centroid that draws the log-to-model relationships. This work is detailed in Chapter 4.

Anti-alignment: Chapter 5 tackles anti-alignment for measuring the precision of process model. Anti-alignments are the most deviant runs of process model with respect to the log traces. This artefact focuses on another direction of conformance checking, the idea is to get the extra modeled behaviors.

We motivated the use of those conformance checking artefacts for log trace clustering in order to extract relevant model-based trace variants. Thanks to process discovery and the quality criteria of conformance checking, a process owner can learn and validate a process model that designs its process. Once a model is accepted, it can be used as a reliable baseline for partitioning the recorded behaviors contained in log. The use of formal techniques in our clustering algorithms allow to preserve the conformance of the input process model. Therefore, our model-based clustering methods extract well-defined modeled centroids with the discovered clusters such that the relationships between them have been formally specified. The centroids are either full runs, processes or subnets of the initial model which enables representing respectively causality, choice and concurrency, and loop behaviors. Thus, if the input process model is complex, which is usually the case for good fitting process models, our approach extracts smaller parts of it. The discovered clusters can then be presented for decision making with the centroids, a.k.a., model-based trace variants, as an explainable output.

Furthermore, this thesis contains several case studies that draw examples of application of the methods. About multi-alignment, we retain that this artefact fits well when the set of log traces are similar. Otherwise, the clustering approaches allow to separate the groups of different behaviors in clusters. Experimentation on clustering reveals the importance of the quality of the process model. Finally, we experimented anti-alignment for measuring precision of process models but also for getting an explainable artefact of model deviations.

The important contributions of this thesis is the computation of multi-alignment and anti-alignment which was a gap in the literature, and the development of the model-based clustering of log traces which incorporates choice, concurrency and loops. The techniques developed in those works involve the computation of edit distance between many sequences and the given process model. It places the methods in NP-complete problems.

6.2 Open Issues

The general focus of this thesis is to provide formal definitions and applicable algorithms to the conformance checking artefacts in the context of log trace clustering in process mining. However, despite the proofs of concept of the methods though several experiments, we believe that more experimental studies should be addressed to present statistical results.

Moreover, we identified the following open issues:

- **SAT optimization:** we believe that some engineering of the SAT formulas can reduce or partition their memory need, thus allowing to compute it on most of today's computers. This improvement would enable to get the clustering of the log traces without using the sampling algorithm which disables the centric notion necessary to obtain good clustering.
- **Grouping of optimizations:** the implementation of the A*-based algorithms can be optimized in association with other techniques of the literature like [101, 74]. For instance, the framework of [74] is a divide-and-conquer approach that focus on a certain set of activities to relate the log and model. Thus, an association of our method in their framework is promising for a gain in computation time. More generally, the main idea behind our approximating approach is the notion of prefix first order in processes. This can be applied to windows of the input sequences like in [73, 131].
- **Process model fitness with discounted alignment:** we presented a technique for computing approximation of alignment. Because this artefact is usually used in fitness measure, an open issue of this work is to evaluate process model fitness with these alignment approximations. This addition would enable to compare the quality of the fitness obtained with this contribution and the one given by token replay approach of Berti et al. in [14].
- **Precising quality criteria implementation of model-based clustering:** in this thesis, we elaborated formal definitions of good criteria for getting model-based clustering like the inter-cluster distance and the intra-cluster distance. In practice, we reduced some problems to simple heuristics. For instance, for the intra-cluster distance, we assume that limiting the number of transitions in centroid gives less runs in them. Because the intra-cluster distance is used to minimize the maximal distance between runs of a centroid in order to obtain more precise representatives, limiting the number of runs in centroids reduces this distance and promises better results. Properly, this metric should be incorporating by computing the distance between all runs in all centroids which involves a huge use of edit distance.
- **Data perspective:** event logs contain many information about processes which are not limited to the sequences of activities. Incorporating this knowledge in the

approaches can enriched the result and its explainability. This idea has already been presented in [44] where resource attributes are used to discover process models. By adding data perspective to our works we can enable data explanation of our model-based trace variants.

This list of open issues is not exhaustive and we welcome any improvement of our approaches. Implementation of all the presented methods are available online.

6.3 Future Works

As future works, we have many avenues in view for extending the different approaches presented in this thesis. We first present our proposals for developing and exploring the main topic of this thesis, the model-based clustering approaches:

- **Centroid analysis:** the analysis of the centroids obtained by our model-based clustering can be developed in order to explain the clustering of the log traces in the line of [46, 40]. All along the thesis, we motivated the use of a process model as a reliable baseline in order to explain the processes. Now that we have a set of methods for extracting artefacts from the model, an analysis step should be involved to extract the knowledge from them. The obtained centroids can be aligned to their associated set of log traces and bring several business keys about the clustered processes.

- **Metrics for model-based clustering:** quality criteria given for good model-based clustering have been introduced in this thesis. They open many perspectives for analyzing and improving the output modeled centroids. The latter is a novelty in process mining and it can be followed by a set of metrics like it appeared in classical data clustering for measuring the quality of the discovered clusters [50, 103]. For instance, the Dunn index given in [50] computes a specific ration between the intra-cluster distance and the inter-cluster distance to draw the compactness of the clusters. Similar metrics can be developed by incorporating the model-based centroids. These future works would enable to compare different clusterings of a same log.

- **Process owner knowledge:** involving process owners in the clustering methods can bring novel directions of research behind the same idea of [41, 42] where the expert knowledge is directly incorporated in the algorithm. In those works, a set of must-link and cannot-link constraints is given. The same idea can be easily added to our SAT encoding.

About anti-alignments, we see potential for derivative notions of this artefact:

- **Anti-alignment between nets:** anti-alignments can be generalized between two nets to find behavior of the former that cannot be found in the latter. The found difference is then a run of one of the input models and can be used to compare process models.

- **Anti-alignment extension:** like for the clustering approaches introduced in this thesis, anti-alignment can be extended to processes or subnets in order to find larger part of the model that is deviating with respect to the log.

These two ideas can also be applied for multi-alignment in the opposite perspectives. For instance, multi-alignment between two nets can highlight similarities between process models. Getting many multi-alignments can then draw a certain closeness between these models.

We end this conclusion chapter with another direction of process mining by bouncing back to model repair. Investigation on anti-alignments for removing deviation of process models is certainly the future work of this thesis. Indeed, anti-alignment computation enables a good interpretation of imprecision of process models such that the practitioner can work hand-in-hand with the process owner. We presented a framework for this purpose where the process owner validates or not anti-alignments in a iterative work with the practitioner. Then the latter can repair the process model with the knowledge of correct deviating runs. However, the repairing steps required for applying the framework are still an open issue of this idea.

Chapter 7

French Summary

7.1 Introduction

La quantité d'informations produites, copiées et consommées croît chaque année due à l'expansion des avancées technologiques. On estime un total de 59 zettabytes de données consommées à la fin de l'année 2020 qui continuerait d'augmenter pour atteindre 165 zettabytes en 2025 [1]. La croissance s'est accelérée avec la pandémie du SARS-CoV-2 qui pousse à l'utilisation des moyens de communications digitalisés.

L'analyse des données s'est alors rapidement développée au sein des entreprises qui souhaitent améliorer leur fonctionnement. De nombreux études montrent qu'une mauvaise analyse de données entraîne des coûts décisionnels très importants et cela pour la plupart des domaines [66, 98, 2]. Nous pouvons citer par exemple les hopitaux et les industries qui regroupent un grand nombre d'acteurs et d'actions [106].

Dans cette thèse, nous étudions les données d'évènements enregistrées dans les logs. Structurées comme une collection d'événements, ces données décrivent les processus qui apparaissent dans les organisations. La Fig. 7.1 présente un exemple artificiel d'un journal d'évènements, aussi appelé *log*. Chaque ligne représente un événement. Les techniques classiques d'exploration de données évaluent ces données en travaillant sur les caractéristiques

Case Identifier	Timestamp	Activity	Resource 1	Resource 2
239U	16-03-2020:11.02	start (s)	Proc.1	Silent
187V	16-03-2020:11.07	start (s)	Proc.2	Silent
187V	16-03-2020:11.09	open file (f)	Proc.2	Silent
239U	16-03-2020:11.31	open file (f)	Proc.1	Silent
239U	16-03-2020:12.20	activation (a)	Proc.2	Silent
187V	16-03-2020:12.20	activation (a)	Proc.2	Console
...	...	...	...	...
982R	17-03-2021:12.20	bad request (b)	Proc.2	Console

Figure 7.1: Exemple de log

Case Identifier	Sequence of Activities
239U	$\langle s, f, a \rangle$
187V	$\langle s, f, a \rangle$
...	...
982R	$\langle s, c, b \rangle$

Figure 7.2: Schématique vue séquentielle de données d'évènements

données généralement par les dimensions des données, c'est-à-dire les colonnes. Ces caractéristiques sont ensuite utilisées pour regrouper et exploiter les événements. Cependant, l'ordre des activités dans le temps est une information clé dans l'analyse des processus et cet aspect n'est pas donné par les événements eux-mêmes. En regroupant les événements par identifiant et en les ordonnant par leur timestamp, nous obtenons les informations comportementales comme le montre la Fig. 7.2.

L'objectif suivant est alors de découvrir des comportements dans ces séquences d'activités. On cherche notemment des relations de *causalité* entre les actions, de la *concurrence*, les *choix* possibles et les *boucles*. Cette analyse s'avère rapidement complexe en raison de la quantité de comportements différents contenus dans les logs [76].

Dans la littérature scientifique, il existe différents domaines qui traitent les données séquentielles. Par exemple, le Sequences Pattern Mining se concentre sur les schémas particuliers [59]. Les approches de Deep Learning ont montré d'énormes améliorations dans l'analyse des séquences comme on peut le voir dans le Traitement du Langage Naturel, où les phrases sont des séquences de mots [94]. Le contexte et l'ordre des éléments dans les séquences sont d'un grand intérêt pour la recherche. Cependant, les domaines susmentionnés ne fournissent pas de modèles compréhensibles de bout en bout des séquences étudiées, ce qui est la source principale de l'explicabilité requise dans la gestion des processus d'entreprise, où les prises de décision doivent être fiables. Les modèles de processus se sont avérés être un élément clé pour décrire, analyser et optimiser l'exécution des processus [100]. C'est le Process Mining qui comble le fossé entre le Business Process Management et la Data Science en fournissant un ensemble de techniques pour découvrir et vérifier les modèles de processus [124].

7.1.1 Process mining

Process Mining est un domaine récent qui permet d'extraire des informations des journaux d'événements, communément appelés logs, en produisant des analyses centrées sur les modèles de processus.

Comme vu précedemment, les logs peuvent être réduits à des séquences d'activités en omettant les caractéristiques des évènements comme l'attribut Resource 1 dans la Fig. 7.1. On définit alors les *traces* des journaux comme telles:

Definition 27 (Log, Traces)**.** *Soit* Σ *un ensemble d'activités. On définit un log est un multiset fini de sequences* $\sigma \in \Sigma^*$ *qui sont appelées des* traces. log traces.

Le Process Mining apporte un ensemble d'algorithmes qui permettent aux experts des données d'extraire facilement des modèles de processus représentant les opérations qui structurent leurs organisations. Ces méthodes automatisées de découverte de processus prennent en entrée un journal d'événements et produisent un modèle de processus qui capture les relations entre les tâches observées dans le journal. Les modèles produits sont généralement des réseaux de Petri ou des modèles BPMN (Business Process Management Notation) parce qu'ils permettent de décrire formellement la causalité, la concurrence, le choix et le comportement des boucles [9]. Nous présentons formellement les réseaux de Petri.

Definition 28 (Modèle de Processus (Réseau de Petri Étiqueté) [87])**.** *Un modèle de Processus défini par un* réseau de Petri labelé *est un tuple* $N = \langle P, T, F, m_0, m_f, \Sigma, \Lambda \rangle$ *où* P *est un ensemble de places,* T *un ensemble de transitions (avec* $P \cap T = \emptyset$*),* $F \subseteq (P \times T) \cup (T \times P)$ *les relations,* m_0 *le marquage initial,* m_f *le marquage final,* Σ *l'ensemble des activités donné et* $\Lambda : T \to \Sigma \cup \{\tau\}$ *la fonction de correspondance entre les transitions et les labels où* τ *désigne une activité silencieuse.*

Sémantique. Un marquage correspond à l'ensemble des places qui contiennent un jeton à un instant donné. Une transition peut s'executer lorsque l'ensemble des places en amont contiennent un jeton. Ceci est noté par ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$ où x est la transition que l'on souhaite executer. Les jetons sont alors supprimés de ces places et toutes les places situées en aval de cette transition recoivent un jeton: $x^\bullet \stackrel{\text{def}}{=} \{y \in P \mid (y, x) \in F\}$. On dit qu'un marquage m' est *atteignable* depuis m lorsqu'il existe une séquence de transitions $\langle t_1 \dots t_n \rangle$ qui transforme m en m'. On note cette transformation par $m[t_1 \dots t_n \rangle m'$. L'ensemble des marquage atteignable depuis m dans N est noté $RS(N, m)$.

En Process Mining, les transitions représentent les activités et les places correspondent aux états des processus.

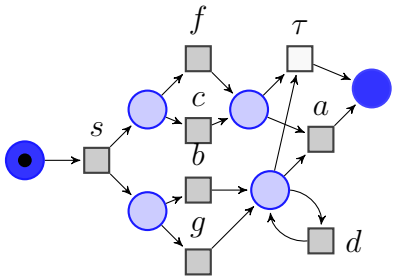


Figure 7.3: Un model N

Il existe différentes classes de réseaux de Petri. Ici nous utilisons les réseaux de Petri sauf et plutôt-fiable, c'est-à-dire qu'un seul jeton est contenu dans une même place à un instant donné et qu'il existe au moins une séquence de transitions reliant le marquage initial au marquage final. Il est courant en Process Mining que ces réseaux soient contraints à une place maximum pour chacun de ces deux marquages délimitant les processus.

Nous montrons ci-dessus un exemple d'un modèle de processus pouvant représenter les données de la Fig 7.2. On remarque le modèle de processus est capable de représenter certaines séquences du log. Cependant, d'autres ne trouvent pas leur compte dans ce modèle comme la séquence $\langle s, f, a \rangle$. En effet, le pattern de concurence, fermé par a ici, nécessite l'activation d'une des transitions b ou g pour cette séquence. C'est là qu'intervient la *Vérification de Conformité*, *Conformance Checking* en anglais, un sous-domaine du Process Mining qui vise à vérifier les correspondances entre les informations contenues dans le log et celles modélisées.

7.1.2 Vérification de conformité

La Vérification de Conformité, a.k.a, Conformance Checking, vise à comparer les comportements décrits par le modèle et ceux enregistrés dans les logs. Pour cela, quatre critères ont été définis, à savoir:

- **Adéquation:** qui vérifie que les comportements observés sont bien présents dans le modèle.
- **Précision:** qui présente la quantité modélisée qui est bien contenue dans les logs.
- **Généralisation:** qui représente la capacité du modèle à capturer des comportements pas encore observés dans le log mais bien présents dans le système.
- **Simplicité:** qui quantifie la densité et l'interprétabilité des modèles.

Un compromis entre ces critères de qualité est un grand dilemme du domaine en raison de la grande complexité des données d'évènements [120].

7.1.3 Variantes de traces et méthodes de partitionnement

Pour réduire la complexité des logs qui induisent la découverte de modèles de processus complexes, plusieurs méthodes ont été développées. Par exemple, de nombreux travaux proposent d'étudier les processus en analysant les *variantes* de traces, c'est-à-dire, les séquences uniques d'activités [114]. Par exemple, on observe en Fig. 7.2 deux fois le comportement $\langle s, f, a \rangle$ qui peuvent donc être représentés par une seule même séquence que l'on appelle variante.

Par ailleurs, une autre méthode permettant de regrouper les traces d'un log est le *partitionnement*, *clustering* en anglais. Le partitionnement est la tâche qui consiste à répartir les objets en différents groupes appelés clusters, dans lesquels les objets sont similaires. Le regroupement d'instances de processus est donc la partition des instances de journaux en sous-journaux, aussi appelés *sublogs*, de sorte que ces groupes rassemblent des processus similaires. Ce sujet de recherche a suscité un grand intérêt pour l'exploration de processus au cours des deux dernières décennies, avec 103 travaux pertinents d'après une récente

étude [144]. Au final, la similarité des instances de processus a été abordée sous plusieurs angles :

- D'une part, l'étude du *flux* donné par les traces permet de regrouper les instances de processus en fonction du comportement qu'elles décrivent. En d'autres termes, on évalue les activités qui apparaissent dans le système. Ces méthodes de regroupement vont de l'étude de la fréquence des activités [108] à l'étude des patterns [63, 39, 21, 78].
- D'autre part, des approches traitent le *contexte* des évènements et fournissent un regroupement basé sur les attributs de données tels que "Ressource 1" et "Ressource 2" de la Fig. 7.1. Ces techniques se rapprochent de l'exploration de données habituelle [133].
- Certains travaux emploient des deux approches combinées [108, 79].

Aucun des travaux énoncés précédemment ne considèrent l'existance d'un modèle de processus pour regrouper les instance de processus. Les variantes de traces et différentes approches de partitionnement des traces sont uniquement basées sur les données contenues dans les logs. Dans cette thèse, nous proposons de combler ces lacunes de la littérature en proposant des rapproches permettant d'extraire des variantes modélisées des traces grâce, entre autre, à des méthodes de clustering.

7.1.4 Problématique et motivation

Une fois qu'un modèle de processus a été validé par son propriétaire, l'expert des données peut bénéficier de l'existence de ce modèle en l'utilisant comme base de référence pour l'analyse des logs. Ainsi, les algorithmes d'extraction de variantes de traces et le partitionnement d'instances de processus peuvent utiliser ce modèle comme entrée. Cette idée s'oppose légèrement à l'idée de découvrir des modèles plus simples en partionnant les traces au préalable. Ici, le modèle de processus, qui représente le log entier, peut être complexe et l'objectif est d'en extraire des artefacts plus simples. Cette perspective se motive par la complexité des modèles de processus produits par les algorithmes de découverte qui priorétisent habituellement à l'adéquation [120]. Puisque le modèle appris contient les informations des processus et donne une visualisation de ceux-ci, une analyse de logs basée sur le modèle apporte une vue nouvelle pour la prise de décision.

Dans cette thèse, nous présentons plusieurs approches qui utilisent des techniques de Vérification de Conformité pour représenter les sous-ensembles de log en se basant sur un modèle de processus. Ces sous-ensembles sont alors représentés par des artefacts extraits du modèle que nous utilisons comme *variantes modélisées de traces*.

7.1.5 Contributions et plan du résumé en langue française

Ce présent rapport est un résumé en langue française qui accompagne la thèse intitulée *Process Instance Clustering Based on Conformance Checking Artefacts*, elle, rédigée

en anglais. Les contributions de cette thèse se concentrent sur les trois grands artefacts de la Vérification de Conformité, à savoir les *alignements*, les *multi-alignements* et les *anti-alignements*:

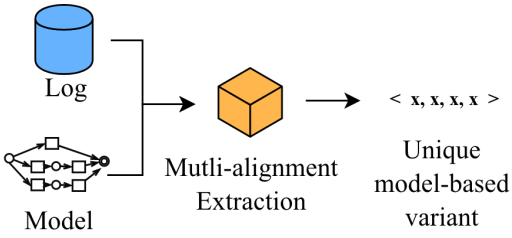


Figure 7.4: Extraction de multi-alignements

Les *multi-alignements* sont une première approche de variantes modélisées de traces. A partir d'un ensemble d'instances, une unique séquence médiane est extraite du modèle pour représenter les comportements donnés en entrée. On remarque que les multi-alignements sont inappropriés lorsque les traces font référence à différents types de comportements où plusieurs séquences seraient nécessaires pour les représenter.

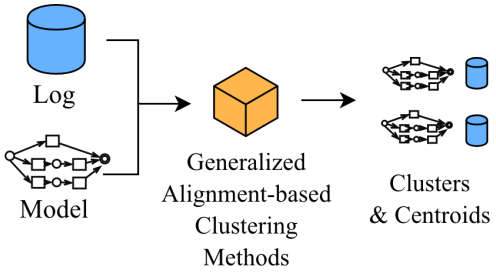


Figure 7.5: Partitionnement des traces en utilisant les alignements

Nous utilisons alors les *alignements* dans des méthodes de partitionnement qui, à partir d'un log et d'un modèle de processus, extraient des clusters associés à des variantes modélisées des différents groupes des traces qui ont été automatiquement identifés. Nous proposons trois types de partitionnement des traces dont les variantes modélisées, qui font office de centroids, ont des structures différentes. Elles sont alors soit une séquence, un processus ou un sous-modèle, tous extraits du modèle initial. Ces trois algorithmes sont liés par des propriétés des réseaux de Petri que nous détaillons dans la thèse.

Enfin, l'utilisation du modèle en entrée de nos méthodes fait réfléchir sur la pertinence de celui-ci. C'est dans cette dernière perspective que nous élaborons les *anti-alignements*. Définis comme les séquences du modèle les plus éloignées des traces du log, les anti-alignements permettent de calculer la précision du modèle.

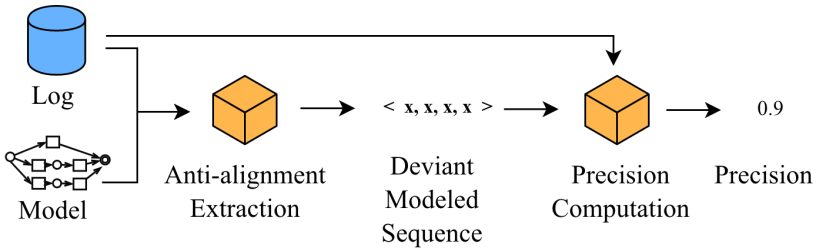


Figure 7.6: Calcul de la précision avec les anti-alignements

7.2 Les artefacts de vérification de conformité

Un modèle de processus permet de représenter un ensemble de processus différents. Cependant le langage d'un tel modèle est rarement celui contenu dans le log à cause de la complexité des données d'évènements. Les relations log-modèle peuvent alors s'établir grâce aux artefacts de la Vérification de Conformité.

7.2.1 Les alignements

Nous présentons d'abord les alignements qui ne sont pas une nouveauté en Process Mining [4].

Definition 29 (Alignement)**.** *Pour une trace de log σ et un modèle de processus N, un* alignement *(optimal) de σ dans N est une séquence entière générée par N, $u \in Runs(N)$, telle que la distance $dist(\sigma, u)$ entre cette séquence u et la trace σ soit minimale:*

$$\min_{u \in Runs(N)} dist(\sigma, u) \tag{7.1}$$

où dist fait référence à une distance entre séquences.

En Process Mining, la distance usuellement utilisée est la distance de Levenshtein que nous rappelons.

Definition 30 (Distance de Levenshtein)**.** *La* distance de Levenshtein $\mathcal{L}(u, v)$ *entre deux sequences u and $v \in \Sigma^*$ est le minimum d'éditions nécessaires pour transformer u en v. Dans notre cas, les éditions peuvent être des suppressions ou des additions d'activités dans les séquences.*

Example 7.2.1. *Un alignement de $\sigma = \langle s, f, a \rangle$ dans N est $\langle s, f, b, a \rangle$. La distance entre ces deux séquences est alors de 1 pour la suppression de b.*

On note que les alignements sont l'extraction d'une séquence par trace de log. Or, un journal d'évènement contient habituellement un grand nombre de traces. Obtenir un alignement par trace nuit à la lisibilité.

7.2.2 Les multi-alignements

Pour extraire une unique séquence du modèle qui représente plusieurs séquences de log, on introduit les multi-alignements.

Definition 31 (Multi-alignement)**.** *Pour un ensemble fini de traces de log L et un modèle de processus N, un* multi-alignement *(optimal) de L dans N est une séquence entière de N, $u \in Runs(N)$, telle que la distance maximale de u à $\sigma \in L$ est minimale:*

$$\min_{u \in Runs(N)} \max_{\sigma \in L} dist(\sigma, u) \tag{7.2}$$

où dist est une distance entre séquences, communément la distance de Levenshtein.

Example 7.2.2. *Pour l'ensemble $L_1 = \{\langle s, f, a\rangle, \langle s, f, b, a\rangle, \langle s, b, b, a\rangle\}$ de traces de log, un multi-alignement de L_1 dans N est $\langle s, f, b, a\rangle$ qui est à une distance maximale de 2 pour toutes les traces de L_1.*

7.2.3 Les anti-alignements

Les anti-alignements donnent l'opposé des multi-alignements. Un anti-alignement est une séquence qui existe dans le modèle de processus alors qu'elle s'éloigne particulièrement des traces du log.

Definition 32 (Anti-alignement)**.** *Pour un ensemble fini de traces de log L et un modèle de processus N, un* anti-alignement *(optimal) de L dans N est une séquence entière de N, $u \in Runs(N)$, telle que la distance minimale de u à $\sigma \in L$ est maximale:*

$$\max_{u \in Runs(N)} \min_{\sigma \in L} dist(\sigma, u) \tag{7.3}$$

où dist est une distance entre séquences, communément la distance de Levenshtein.

A cause des boucles, les anti-alignements peuvent s'éloigner indéfiniment des traces. Pour résoudre ce problème, on peut normaliser la distance sur la taille de l'anti-alignement ou lui fixer une taille de manière arbitraire.

Example 7.2.3. *Pour L_1 donné en Exemple 2.2.1, un anti-alignement de taille 8 dans N est $\langle s, c, g, d, d, d, d, \tau\rangle$ qui est à une distance minimale de 8 de toutes des traces de L_1.*

7.3 Variantes modélisées de traces

L'intérêt principal de la thèse résumée est d'extraire des sous parties du modèle de processus qui représentent les traces. Voyons dans cette partie les différentes étapes qui permettent d'obtenir ces arteafacts que nous appelons *variantes modélisées de traces.*

7.3.1 Les multi-alignements comme variantes modélisées de traces et ses limites

Comme nous l'avons vu précedemment, les multi-alignements donnent une représentation médiane modélisée des traces données. Ainsi, un multi-alignement de traces similaires donne une bonne variante modélisée des traces. Cependant pour des traces distinctes, les multi-alignements s'avèrent inadaptés. En effet, en minimisant la distance maximale aux traces, les multi-alignements cherchent à faire un compromis entre les différents processus observés ce qui, parfois, donne des séquences éloignées de toutes les traces de log.

Example 7.3.1. *Pour* $L_2 = \{\langle s, f, b, a\rangle, \langle s, c, g, d, d, d, d\rangle\}$*, un multi-alignement optimal est* $\langle s, f, g, d\rangle$ *à distance minimale de* 4 *des traces et ne représente pas vraiment les traces.*

7.3.2 Méthodes de partitionnement de traces basées sur les alignements

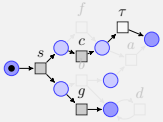
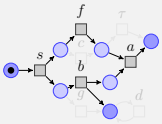
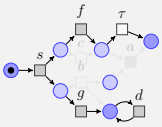
Le partitionnement des traces basé sur un modèle de processus est la contribution la plus importante de cette thèse. Les traces de logs sont regroupées en considérant le modèle comme une base de référence. Des parties des modèles, appelées *centroides*, représentent les clusters et peuvent alors être utilisés comme variantes modélisées des traces. Nous proposons trois types de méthodes de partitionnement dont la différence est le type de centroides acceptés:

- *Partitionnement basé sur les alignements* : cette version a été introduite dans [33]. Les centroids sont des séquences du modèle qui permettent un bon partitionnement des traces à l'aide de multi-alignements.

- *Partitionnement basé sur les alignements et les ordres partiels* : pour accepter les choix et la concurence d'activités dans les centroids, nous avons élaboré cette deuxième méthode de partitionnement qui utilise des ordres partiels du modèle initial comme centroids. Ainsi, un centroid contient plusieurs séquences auxquelles sont alignées les traces.

- *Partitionnement basé sur les alignements et les sous-modèles* : l'incorporation des boucles s'est fait en acceptant des sous-modèles du modèle initial comme centroids.

Example 7.3.2. *Nous donnons ci-après un exemple de partitionnement de 7 traces basé sur les alignements et les sous-modèles (méthode nommée AMSTC en anglais).*

Cluster 1	*Cluster* 2	*Cluster* 3
		
$\langle s, c, g \rangle$	$\langle s, b, f, a \rangle$	$\langle s, g, f, d, d \rangle$
$\langle s, c, g, d \rangle$	$\langle s, f, b, a \rangle$	$\langle s, g, f, d, d, d, d \rangle$
	$\langle s, f, f, a \rangle$	

Pour obtenir des partitionnements qualitatifs, nous introduisons dans la thèse un ensemble de critères comme la distance maximale entre une trace et son centroide.

7.3.3 Impact de la qualité des modèles

Les variantes modélisées des traces étant extraites du modèle initial, sa qualité est un critère important pour obtenir de bonnes variantes capables de représenter aux mieux les traces qui leurs sont associées. L'adéquation du modèle se mesure à l'aide d'alignements entre les traces et le modèle. Cette méthode fait l'unanimité au sein de la communauté. Ce n'est pas le cas du calcul de la précision qui peine à accorder tous les chercheurs du domaine. La précision doit répondre à cinq axiomes précis donné par [109]. Nous proposons un calcul de la précision, qui vérifie ces axiomes, à l'aide des anti-alignements:

Definition 33 (Précision basée sur les anti-alignements)**.** *Soit L un journaux d'évènements et N un modèle de processus. La précision de N par rapport à L est:*

$$P^{\epsilon}_{aa}(N, L) \stackrel{\text{def}}{=} 1 - \sup_{\gamma \in Runs(N)} \min_{\sigma \in L} \frac{\Delta(\gamma, \sigma)}{(1+\epsilon)^{|\gamma|}} \tag{7.4}$$

où $\Delta(\gamma, \sigma) = \frac{dist(\gamma,\sigma)}{|\gamma|+|\sigma|}$, γ étant un anti-alignement, et $\epsilon \geq 0$.

7.4 Implémentation des méthodes

Les fondements des multi-alignements et anti-alignements sont présentés dans la thèse et, pour la première fois, leur calcul exact est obtenu grâce à des algorithmes optimaux. Ce codage ayant des limites en espace de mémoire, nous présentons une seconde méthode qui fournit des approximations de ces artefacts grâce à une nouvelle distance escomptée que nous introduisons.

7.4.1 Encodage SAT

Le problème de satisfiabilité (SAT), est le problème qui consiste à déterminer, pour une formule booléenne donnée, s'il existe une combinaison d'affectations aux variables qui la satisfait. En d'autres termes, pour l'alignement, le multi-alignement et l'anti-alignement, nous définissons des problèmes de satisfiabilité que nous codons en formules booléennes :

- *Alignement* : pour une traceσ et un modèle N, existe-t-il une sequence u de N telle que $dist(\sigma, u) \leq d$? (où $dist$ est la distance de Levenshtein et $d \in \mathbb{N}$)
- *Multi-alignement* : pour un log L et un modèle N, existe-t-il une sequence u de N telle que $\forall_{\sigma \in L} dist(\sigma, u) \leq d$?
- *Anti-alignement* : pour un log L et un modèle N, existe-t-il une sequence u de N telle que $\forall_{\sigma \in L} dist(\sigma, u) \geq d$?

De plus, les problèmes exacts consistent à obtenir une distance optimale. Par exemple, pour l'alignement, au lieu de chercher une distance inférieure ou égale à d, la définition vise à obtenir la distance minimale, c'est-à-dire quelle séquence du modèle minimise la distance à la trace. La minimisation peut être encodée dans des problèmes MaxSAT qui fonctionnent avec des clauses pondérées.

Les méthodes de partitionnement sont elles aussi implémentées en formules satisfiables à partir de l'idée suivante:

- *Partitionnement des traces* : pour un log L et un modèle N, existe-t-il un partitionnement des traces basé sur un modèle vérifiant X ? où X mentionne les critères de qualité d'un partitionnement qui n'ont pas été détaillés dans ce résumé.

Les formules sont composées de trois parties: la sémantique d'un réseau de Petri, la distance d'édition et la minimisation ou la maximisation des distances. Les trois artefacts de Vérification de Conformité utilise ces trois mêmes blocs ce qui permet rapidement de passer d'un codage à l'autre. Toutes les variables et les formules sont détaillées dans la thèse.

7.4.2 Distance escomptée

La plupart des algorithmes de calcul d'alignement en Process Mining utilisent une recherche du plus court chemin dans un graphe appelé le *produit synchrone* liant le modèle de processus et la trace de log donnée [4]. Les algorithmes sont notamment Dijsktra et A*. Cependant pour des modèles complexes, parcourir ce graphe est une tâche difficile et étendre l'idée aux multi-alignements et aux anti-alignements devient alors impossible. Pour réduire le nombre de chemins à parcourir nous introduisons une nouvelle distance que nous nommons la *distance escomptée*, *discounted edit distance* en anglais. L'idée de cette distance est de priorétiser les alignements d'activités en début de processus, supprimant ainsi tous les chemins du modèle qui s'éloignent de la trace dès le début.

Definition 34 (Distance escomptée)**.** *Nous définissons la* distance escomptée *entre deux séquences u et v avec le paramètre $\theta \geq 1$ par $\mathcal{D}_\theta(u,v) \stackrel{\text{def}}{=} \mathcal{D}^0_\theta(u,v)$ où:*

$$\mathcal{D}^k_\theta(u,v) = \begin{cases} \mathcal{D}^k_\theta(\langle\rangle,\langle\rangle) = 0 \\ \mathcal{D}^k_\theta(\langle\rangle, b.v') = \mathcal{D}^{k+1}_\theta(\langle\rangle, v') + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u',\langle\rangle) = \mathcal{D}^{k+1}_\theta(u',\langle\rangle) + \theta^{-k} \\ \mathcal{D}^k_\theta(a.u', b.v') = \mathcal{D}^{k+2}_\theta(u',v') \qquad si\ (a == b) \\ \mathcal{D}^k_\theta(a.u', b.v') = \min \begin{cases} \mathcal{D}^{k+1}_\theta(u',v) + \theta^{-k} \\ \mathcal{D}^{k+1}_\theta(u,v') + \theta^{-k} \end{cases} \quad autrement. \end{cases} \tag{7.5}$$

Ainsi, insertions et suppressions coûtent θ^{-k} où k fait référence à la position de l'édition.

Cette nouvelle distance est proche de la distance de Levenshtein. On note par exemple que pour $\theta = 1$, on obtient la distance de Levenshtein.

Example 7.4.1. *Let $u = \langle x,a,b\rangle$ and $v = \langle a,y,b\rangle$. The discounted edit distance between u and v is $\mathcal{D}^{u,v}_\theta = \theta^{-1} + \theta^{-4}$. If $\theta = 1$, the distance equals to 2 and is the Levenshtein edit distance where deleting x costs 1 and adding y costs 1.*

On montre dans la thèse comment, en utilisant cette distance escomptée, on réduit considérable le nombre de séquences du modèle pouvant être un bon alignement, multi-alignement ou anti-alignement selon l'algorithme étudié.

7.5 Expériences

Les contributions de cette thèse ont été implémentées au sein de trois logiciels:

- DarkSider : DarkSider est le premier outil qui a implémenté les méthodes optimales. Démarré par Thomas Chatain en 2016, le logiciel est implémenté en Ocaml, un langage fonctionnel qui permet une bonne typographie utile pour l'encodage de formules SAT. L'outil est un logiciel en ligne de commande qui nécessite uniquement un terminal.

- da4py : En raison du fort intérêt pour Python dans la communauté d'analyse de données, da4py est d'abord une traduction de DarkSider en Python 3. Puis, le codage SAT des méthodes devenu plus performant dans cette version Python que dans l'outil en Ocaml grâce à la bibliothèque pysat qui fait le pont entre Python et toutes les différentes sorties des solveurs SAT [68].

- pm4py : Enfin, les dernières contributions de la thèse, à savoir l'introduction de la distance escomptée qui s'associe à un algorithme par artefact, sont implémentées dans une copie de pm4py dans l'objectif d'étendre la disponibilité des artefacts de Vérification de Conformité à la communauté. On notera que ces versions sont des

approximations des artefacts qui permettent d'avoir des résultats rapides sur des données réelles où les versions optimales ne permettent pas toujours d'obtenir des résultats.

La thèse contient un ensemble d'expériences qui servent de preuves de concept des méthodes introduites. On présente ces résutats pour des jeux de données artificiels mais aussi des jeux de données réels. Dans la plupart des cas, ces données sont connues de la communauté. On peut par exemple citer les logs des Bussiness Process Management Challenge [127]. Enfin, la thèse contient, en annexe, des tutoriels pour utiliser les différentes méthodes implémentées.

7.6 Conclusion

Cette thèse présente une étude des artefacts de Vérification de Conformité dans le contexte du partitionnement des traces de logs. Le thème principal est la relation entre les modèles de processus et les journaux d'évènements qui est donnée par ces artefacts. Nous avons étudié les *alignments*, *multi-alignments* et *anti-alignments* dans l'objectif d'extraire de bons représentants des instances de processus que nous appelons *variantes modélisées de traces*. La thèse fournit des définitions formelles de nos approches associées à des algorithmes optimaux qui permettent de présenter des preuves de concept des méthodes. Les algorithmes sont encodées en problèmes SAT, ce qui contribue à une récente famille de méthodes algorithmiques pour la Vérification de Conformité [31, 19]. De plus, des heuristiques de ces algorithmes et des algorithmes d'approximation sont développés. Ils permettent de travailler sur de grandes entrées réelles pour une petite perte de qualité. Toutes les méthodes ont été expérimentées et comparées à plusieurs approches de pointe pour des journaux artificiels et réels.

Nous avons identifiés des améliorations possibles de nos travaux que nous listons ci-après :

- **Optimisation des formules SAT :** l'utilisation en mémoire des formules SAT peut sûrement être réduite en améliorant le codage
- **Association de plusieurs optimisations :** nous proposons une heuristique pour réduire l'espace de recherche des alignements qui peut être accumulée avec d'autres techniques de la littérature comme [101, 74]. Par exemple, [74] utilise une méthode de diviser-pour-régner qui se concentre sur des sous-ensemble d'activités. Ainsi, une association de notre méthode dans leur cadre est prometteuse.
- **Calcul de l'adéquation avec nos alignements approximatifs :** nous avons présenté une technique pour calculer des approximations d'alignement. Comme cet artefact est généralement utilisé dans la mesure de l'adéquation, une question ouverte est d'évaluer l'adéquation du modèle de processus avec ces approximations d'alignement. Cette étude permettrait de comparer la qualité de l'adéquation du

mondel obtenue avec cette contribution et celle donnée par l'approche token-replay de Berti et al. in [14].

Enfin, nous voyons un grand nombre de travaux futurs qui pourraient suivre nos contributions :

- **Analyse des Centroids:** l'analyse des centroids obtenus par nos méthodes de partitionnement peut être développée afin d'expliquer les clusters comme dans citede2014secpi,de2017explaining.
- **Mesures de qualité pour les partitionnements basés sur des modèles de processus :** Des critères de qualité donnés pour un bon clustering basé sur un modèle ont été introduits dans cette thèse. Ils ouvrent de nombreuses perspectives pour analyser et améliorer les centroides modélisés obtenus. Ce dernier point est une nouveauté dans le domaine de la fouille de processus et il peut être suivi par un ensemble de mesures comme il est apparu dans le clustering de données classique pour mesurer la qualité des clusters découverts [50, 103]. Par exemple, l'indice de Dunn donné dans [50] calcule un rapport spécifique entre la distance intra-cluster et la distance inter-cluster pour estimer la compacité des clusters. Des métriques similaires peuvent être développées en incorporant les structures particulières des centroids de nos méthodes.
- **Connaissances du propriétaire du processus :** l'implication des propriétaires de processus dans les méthodes de clustering peut apporter de nouvelles directions de recherche derrière la même idée que [41, 42] où la connaissance experte est directement incorporée dans l'algorithme. Dans ces travaux, un ensemble de contraintes "doit-être-lié" et "ne-peut-pas-être-lié" est donné. La même idée peut être facilement ajoutée à notre encodage SAT.
- **multi-alignement et anti-alignement entre modèles :** les multi-alignement et anti-alignements peuvent être généralisés entre deux modèles de processus pour trouver un comportement du premier qui se rapproche ou s'éloigne le plus des comportements du second.
- **Extension des multi-alignement et anti-alignment :** comme pour les approches de clustering introduites dans cette thèse, les artefacts de Vérification de Conformité peut être étendus aux ordres partiels ou aux sous-modèles afin de trouver la plus grande partie du modèle qui se rapproche, pour les multi-alignements, ou qui est déviante, pour les anti-alignements, du journal d'évènements.

Appendices

Appendices A

Tutorials

The scripts are mainly in Python 3. We use the pm4py library that proposes a large set of techniques and objects for Process Mining.

The following lines show how to load a XES log and a PNML process model.

```
from pm4py.objects.log.importer.xes import importer as xes_importer
from pm4py.objects.petri.importer import importer as petri_importer

# load log and model
log = xes_importer.apply("/examples/medium/model2multi.xes")
net, marking, fmarking =
 ↪  petri_importer.apply("/examples/medium/model2.pnml")
```

A.1 Using the SAT-based Algorithms for Alignments, Multi-alignments and Anti-alignments

The SAT-based algorithm for computing the conformance checking artefacts is given in darksider and da4py. Despite that darksider is out-of-date, we provide below some commands to creates and solve the SAT formulas. The tool only runs for TPN type of files for models and TR files for logs. The minimization of the sum of distances is given for multi-alignments and the maximization of the sum of distance for anti-alignments.

> ./darksider -MA ./data_test/example_anti/LoanA.tpn ./data_test/example_anti/LoanA.tr 10 20

- where -MA stands for ”multi-alignment”, 10 is the size of the run and 20 the allowed number of edits in the SAT formulas.

> ./darksider -AA ./data_test/example_anti/LoanA.tpn ./data_test/example_anti/LoanA.tr 10 22

- here -AA stands for ”anti-alignment”.

The solid and up-to-date version of the SAT implementation for alignment, multi-alignments and anti-alignments is given in da4py. The next box shows how to get a multi-alignment for a given log and model. We give some expected outputs below.

```
from da4py.main.conformanceChecking.conformanceArtefacts import
↪  ConformanceArtefacts

artefacts = ConformanceArtefacts()
# the algorithm is implemented for both the edit and hamming distance
artefacts.setDistance_type("edit")
# classical multi-alignment minimizes the maximal distance
artefacts.setOptimizeSup(False)
# the algorithm limits the search of unbounded models
artefacts.setSize_of_run(8)
# this is an heuristic to reduce the runtime, optimal is size_of_run*2
artefacts.setMax_d(16)

# run a multi-Alignment
artefacts.multiAlignment(net, marking, fmarking, log)

# print the artefact
print(artefacts.getRun())
# show the distance between the traces and the artefact
print(artefacts.getTracesWithDistances())
```

> ['S', 'C', 'B', 'tau', 'w', 'w', 'w', 'w']

> distance traces

3 [S, F, B, A]
2 [S, B, F]
1 [S, C, B, A]
4 [S, G, C, A, D]
4 [S, D, D]

The ’w’ hold for ’wait’ activity and appear in order to fill the expected length (8 in this example). See page 31 for more details.

A.2 Using the A*-based Algorithm with the Discounted Edit Distance for Computing the Conformance Checking Artefacts

The A*-based algorithm for computing alignments, multi-alignments and anti-alignments is implemented in pm4py. Three separated packages are involved:

- pm4py.algo.conformance.alignments
- pm4py.algo.conformance.antialignments
- pm4py.algo.conformance.multialignments

The use of the algorithms is based on the existing method contained in the library for computing alignment. We give an example below.

```
from pm4py.algo.conformance.alignments import algorithm as ali

# select the discounted method
algorithm = ali.VERSION_DISCOUNTED_A_STAR

# in this example we chose the simulation of the synchronous product
my_params = {ali.Parameters.SYNCHRONOUS:False, ali.Parameters.EXPONENT:1.5}

# play the alignments
alignments = ali.apply(log, net, marking, fmarking, variant=algorithm,
↪  parameters=my_params)

print(alignments)
```

A.3 AMSTC

The AMSTC algorithm uses subnets as centroids. In the next script box we detail the different settings.

```
# process model
model, m0, mf = importer.pnml.import_net('examples/medium/model2.pnml')

# log traces
traces = xes_importer.import_log('examples/medium/model2.xes')

# sampleSize : number of traces that are used in the sampling method
sampleSize= 5
```

```

# sizeOfRun : maximal length requested to compute alignment
sizeOfRun = 8

# maxNbC : maximal number of transitions per cluster to avoid to get a unique centroid
maxNbC = 5

# m : number of cluster that will be searching at each AMSTC of the sampling method. Understand that more than m cluster can be returned.
m = 2

# maxCounter : as this is a sampling method, maxCounter is the number of fails of AMSTC before the sampling method stops
# silent_label : every transition that contains this string will not cost in alignment
clustering = samplingVariantsForAmstc(net, m0, mf, log, sampleSize, sizeOfRun, maxD, maxNbC, m, maxCounter=1, silent_label="tau")
```

The clustering output can then be used like :

```
from pm4py.visualization.petrinet import factory as vizu

for (centroid, traces) in clustering:
    if type(centroid) is tuple:
        net, m0,mf=centroid
        vizu.apply(net, m0, mf).view()
        print(traces)
```

Appendices B

Experiment Scripts

In each chapter of the present thesis, we have elaborated a set of experiments to depict the value of our contributions. We give most of the scripts that have been created for this purpose. Some experiments are only a variant of other ones which we have removed for redundancy.

B.1 Comparing Alignments Methods

The following script has been used in Section. 3.5.1 for comparing the A*-based algorithm and its parameter influence with respect to the baseline implemented in pm4py.

```
def execute_script(log_path, pnml_path, version, parameters):
    '''
    This function has been used to compute the differences between
    alignment methods.
    '''
    # load log and model
    net, marking, fmarking = petri_importer.apply(pnml_path)
    log = xes_importer.apply(log_path)

    # get the variants
    variants = get_variants(log)

    # we save asyn moves, time and length of variants
    sum_asyn_moves=[]
    diff_time=[]
    lent=[]

    for i in variants:
        trace = variants[i][0]
        lent.append(len(trace))
```

```
20
21          start = time.time()
22          alignments = ali.apply(trace, net, marking,
            ↪  fmarking,variant=version, parameters=parameters)
23          # save time to run this alignment
24          diff_time.append(time.time()-start)
25
26          # get the number of moves
27          moves = (len(str(alignments["alignment"]).split(">>")))-1
28          sum_asyn_moves.append(moves)
29
30      return str(""+ "{:.2f}".format(sum(sum_asyn_moves) /
        ↪  len(sum_asyn_moves))+"\t"+ "{:.2f}".format(np.std(sum_asyn_moves))+
        ↪  "\t"+"{:.2f}".format(sum(diff_time)))
31
32  def run_exp():
33      '''
34      This function runs the experiments.
35      '''
36      list = ["2012", "2018_pa", "2019", "2020_dd", "2020_rp"]
37      folders = ["im_SI","sm_SI"]
38
39      for f in folders:
40          for log in list:
41           log_path = "./tests/log/"+str(log)+".xes" # check if log are
              ↪  decompressed or add .gz
42              pnml_path = "./tests/model/"+f+"/"+log+".pnml"
43
44              print("\n----", f, log)
45              # execute the state-of-the-art methods
46              print("DSA\t", execute_script(log_path, pnml_path,
                ↪  ali.VERSION_DIJKSTRA_NO_HEURISTICS, {}))
47
48              print("DLMA\t", execute_script(log_path, pnml_path,
                ↪  ali.VERSION_DIJKSTRA_LESS_MEMORY,{}))
49
50              print("DSA\t", execute_script(log_path, pnml_path,
                ↪  ali.VERSION_DIJKSTRA_NO_HEURISTICS, {}))
51              print("DLMA\t", execute_script(log_path, pnml_path,
                ↪  ali.VERSION_DIJKSTRA_LESS_MEMORY, {}))
52
53              # give our results for different values of theta
54              for theta in [1, 1.25, 1.5, 1.75, 2]:
```

```
55                  parameters = {ali.Parameters.SYNCHRONOUS:False,
                    ↪  ali.Parameters.NOTCOMPLETEHEURISTIC:True,
                    ↪  ali.Parameters.EXPONENT:theta}
56
57                  print("DPAexp=",str(theta),"\t", execute_script(log_path,
                    ↪  pnml_path, ali.VERSION_DISCOUNTED_A_STAR, parameters))
58
59                  parameters = {ali.Parameters.SYNCHRONOUS:True,
                    ↪  ali.Parameters.NOTCOMPLETEHEURISTIC:True,
                    ↪  ali.Parameters.EXPONENT:theta}
60
61                  print("DSAexp=",str(theta),"\t", execute_script(log_path,
                    ↪  pnml_path, ali.VERSION_DISCOUNTED_A_STAR, parameters))
62
63  if __name__ == '__main__':
64      run_exp()
```

B.2 Computing Multi-Alignment

Thanks to the proposed algorithms and heuristics of this thesis, multi-alignments though edit distance can be computed for real-life instances. The following script gives the results presented in Section. 3.5.2.

```
1   from da4py.main.conformanceChecking.conformanceArtefacts import
    ↪  ConformanceArtefacts
2
3   from pm4py.objects.petri.importer import importer as petri_importer
4   from pm4py.objects.log.importer.xes import importer as xes_importer
5   from pm4py.algo.conformance.multialignments.variants.discounted_a_star
    ↪  import apply as multi
6   from pm4py.algo.conformance.multialignments.algorithm import Parameters
7
8   if __name__ == '__main__':
9
10      # we work on the models discovered with the split miner and the
        ↪  inductive miner
11      folder = ["sm_SI","im_SI"]
12      loglist = ["2012","2018_pa","2019","2020_dd","2020_rp"]
13      for l in loglist :
14          log_path = "log/" + l + ".xes"
15          log = xes_importer.apply(log_path)
16          log._list = log._list[:50]
17
```

```
        for f in folder:
            pnml_path = "model/" + f + "/" + l + ".pnml"
            net, marking, fmarking = petri_importer.apply(pnml_path)

            # SAT Algorithm
            artefacts = ConformanceArtefacts()
            artefacts.setDistance_type("edit")
            # all silent transitions have been set to "tau".
            artefacts.setSilentLabel("tau")
            artefacts.setOptimizeSup(True)
            artefacts.setSize_of_run(10)
            artefacts.setMax_d(20)

            start = time.time()
            artefacts.multiAlignment(net, marking, fmarking, log)
            print(time.time()-start, artefacts.getRun(),
             ↪  artefacts.getMaxDistanceToRun())

            # A* Algorithm
            start = time.time()
            multi_alignment = multi(log,net,marking,fmarking,
             ↪  parameters={Parameters.EXPONENT:1.5,
             ↪  Parameters.MARKING_LIMIT:200})
            print(time.time()-start, multi_alignment['multi-alignment'],
             ↪  multi_alignment['max_distance_to_log'])
```

From the previous experimentation, we observed that multi-alignments can be very far, in term of edit distance, to the collection of log traces that its should be representing. To improve the quality of these model-based variants, we propose some preprocessing steps that we experiment in the next script.

```
import copy
from pm4py.statistics.variants.log import get as variants_module
from pm4py.objects.log.importer.xes import importer as xes_importer
from pm4py.algo.conformance.multialignments.variants.discounted_a_star
 ↪  import apply as multi
from pm4py.algo.conformance.antialignments.algorithm import Parameters
import numpy as np
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from sklearn.cluster import KMeans
from random import randrange


def encoder(traces):
    '''
```

```
    A small sequences of words to sequences of int encoder where a position in vector corresponds to an activity.
    :param traces: (list of string)
    :return (list of int)
    '''
    int_traces = []
    indexOfActivities = []
    for trace in traces:
        for a in trace:
            if a not in indexOfActivities:
                indexOfActivities.append(a)
    for trace in traces:
        my_list = [0 for i in indexOfActivities]
        for a in trace:
            my_list[indexOfActivities.index(a)]+=1
        # length of trace is also a criterion we want to consider
        my_list.append(len(trace))
        int_traces.append(my_list)
    return int_traces


if __name__ == '__main__':

    log = xes_importer.apply("Loan Application.xes.gz")
    net, marking, fmarking = inductive_miner.apply(log)

    # multi-alignment with entire log
    multi_alignment = multi(log,net,marking,fmarking, parameters = {Parameters.EXPONENT:1.05, Parameters.MARKING_LIMIT:200})
    print("Entire Log:", multi_alignment['max_distance_to_log'], multi_alignment['multi-alignment'])

    for number_of_clusters in [3, 6]:
        # multi-alignment with K-mean as preprocessing step
        variants = variants_module.get_variants(log)
        traces = list(variants.keys())
        int_traces = np.asarray(encoder(traces))
        kmedoids = KMeans(n_clusters = number_of_clusters, random_state=0).fit(int_traces)
        temp = copy.copy(log)
        for c in range(0, number_of_clusters):
            # getting each variant once only is sufficient, we are not working on runtime here.
            temp._list = [variants[i][0] for i in variants if kmedoids.labels_[traces.index(i)]==c]
```

```
            multi_alignment = multi(temp, net, marking, fmarking,
            ↪  parameters={Parameters.EXPONENT:1.05,
            ↪  Parameters.MARKING_LIMIT:200})
            print("For ", number_of_clusters, " clusters, ", c, ":",
            ↪  multi_alignment['max_distance_to_log'],
            ↪  multi_alignment['multi-alignment'])

     # multi-alignment with random selection as preprocessing step
    len_of_log = len(log)
    temp = copy.copy(log)
    for repeat in range (0,10):
        temp._list=[]
        for i in range (0,10):
            random_index = randrange(len_of_log)
            temp._list.append(log._list[random_index])
        multi_alignment = multi(temp, net, marking, fmarking,
        ↪  parameters={Parameters.EXPONENT:1.05,
        ↪  Parameters.MARKING_LIMIT:200})
        print("Random ", repeat, " 10traces :",
        ↪  multi_alignment['max_distance_to_log'],
        ↪  multi_alignment['multi-alignment'])
```

Results are of form:

> For 3 clusters, 0 : 9.0 [tau_1, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Appraise property, Check credit history, Assess loan risk, Assess eligibility, skip_5, skip_8, skip_9, Verify repayment agreement, skip_11]

> For 3 clusters, 1 : 10.0 [tau_1, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Return application back to applicant, Receive updated application, Check application form completeness, Appraise property, Check credit history, Assess loan risk, Assess eligibility, skip_5, skip_8, skip_9, Verify repayment agreement, skip_11]

> For 3 clusters, 2 : 9.0 [tau_1, Check application form completeness, Return application back

to applicant, Receive updated application, Check application form completeness, Appraise property, Check credit history, Assess loan risk, Assess eligibility, skip_5, skip_8, skip_9, Verify repayment agreement, skip_11]

B.3 Computing Anti-Alignment and Precision of Process Model

Find below the experiment script for computing anti-alignment by using the A*-based algorithm.

```
from pm4py.objects.petri.importer import importer as petri_importer
from pm4py.objects.log.importer.xes import importer as xes_importer
from pm4py.algo.conformance.antialignments.variants.discounted_a_star
 ↪  import apply as antii
from pm4py.algo.conformance.antialignments.algorithm import Parameters
from pm4py.visualization.petrinet.visualizer import apply as vizu
import time

if __name__ == '__main__':

    # TABLE 5.3 ----------------------
    EPSILON = 0.01
    log = xes_importer.apply("artificial-1.xes")

    # GENERATIVE MODEL
    net, marking, fmarking = petri_importer.apply("fig1.pnml")
    for theta in [1.1,1.5,2.0]:
        start = time.time()
        resAnti = antii(log,net,marking,fmarking, parameters =
         ↪  {Parameters.EXPONENT : theta,Parameters.EPSILON : EPSILON})
        print(time.time()-start)
        print("1", theta, resAnti['anti-alignment'], "P=",
         ↪  resAnti['precision'])

    # FLOWER MODEL
    net, marking, fmarking = petri_importer.apply("fig3.pnml")
    for theta in  [1.5,2.0]:
        start = time.time()
        resAnti = antii(log,net,marking,fmarking, parameters =
         ↪  {Parameters.EXPONENT : theta,Parameters.EPSILON : EPSILON})
        print(time.time()-start)
        print("3", theta, resAnti['anti-alignment'], "P=",
         ↪  resAnti['precision'])
```

```
29
30      # TABLE 5.4 ----------------------
31      # FLOWER MODEL
32      MU =  100
33      THETA = 1.5
34      for epsilon in [0.01, 0.001]:
35          start = time.time()
36          resAnti = antii(log,net,marking,fmarking, parameters =
            ↪  {Parameters.EXPONENT : THETA, Parameters.EPSILON : epsilon,
            ↪  Parameters.MARKING_LIMIT : MU})
37          print(time.time()-start)
38          print("mu3", THETA, resAnti['anti-alignment'], "P=",
            ↪  resAnti['precision'])
39
40      MU = 5
41      EPSILON = 0.01
42      start = time.time()
43      resAnti = antii(log,net,marking,fmarking, parameters =
        ↪  {Parameters.EXPONENT : THETA, Parameters.EPSILON : EPSILON,
        ↪  Parameters.MARKING_LIMIT : MU})
44      print(time.time()-start)
45      print("mu3", THETA, resAnti['anti-alignment'], "P=",
        ↪  resAnti['precision'])
46
47      # TABLE 5.5  ----------------------
48      THETA = 1.5
49      MU = 10
50      EPSILON = 0.01
51      for i in  [1,2,3,4,5,6,7,8,9,12]:
52          pnml_path = "fig"+str(i)+".pnml"
53          net, marking, fmarking = petri_importer.apply(pnml_path)
54          vizu(net,marking,fmarking).view()
55          start = time.time()
56          resAnti = antii(log,net,marking,fmarking, parameters =
            ↪  {Parameters.EXPONENT : THETA,Parameters.EPSILON : EPSILON,
            ↪  Parameters.MARKING_LIMIT : MU})
57          print(time.time()-start)
58          print(i, resAnti['anti-alignment'], "P=", resAnti['precision'])
59
60      # TABLE 5.7 ----------------------
61      THETA = 2
62      MU = 5
63      EPSILON = 0.01
64      folder = ["sm_SI","im_SI"]
65      log = ["2012", "2019","2020_dd","2020_rp"]
```

```
    for l in log :
        log_path = "./log/"+l+".xes"
        log = xes_importer.apply(log_path)
        for f in folder:
            pnml_path = "./model/"+f+"/"+l+".pnml"
            net, marking, fmarking = petri_importer.apply(pnml_path)
            start = time.time()
            resAnti = antii(log,net,marking,fmarking, parameters =
            ↪  {Parameters.EXPONENT : THETA,Parameters.EPSILON : EPSILON,
            ↪  Parameters.MARKING_LIMIT : MU})
            print(time.time()-start)
            print(l, f, "---",r esAnti)
            print("P=", resAnti['precision'])
```

We computed clustering of some process models to explore potential relationships between conformance criteria and clustering outputs.

```
log = xes_importer.apply("<log_path>")
   net, marking, fmarking = petri_importer.apply("<model_path>")

   alignment = ali.apply(log._list[0], net, marking, fmarking)
   clustering = samplingVariantsForAmstc(net, marking, fmarking, log,5,
   ↪  len(alignment)+5 , 3, len(alignment)+5, 2
   ↪  ,maxCounter=10,editDistance=True,silent_label="tau", debug=None)
   print("2012, im SI A+5; ",len(clustering)-1,sum([len(clustering[i][1])
   ↪  for i in range(len(clustering)-1)]), len(clustering[-1][1]))
```

The next script is used for comparing clustering of some process model of the same log and same process discovery algorithms but different thresholds.

```
from pm4py.algo.discovery.inductive.parameters import Parameters as
↪  inductive_minerp
from pm4py.objects.log.importer.xes import importer as xes_importer
from da4py.main.analytics.amstc import samplingVariantsForAmstc
from pm4py.algo.conformance.alignments import algorithm as ali
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristic_miner
from pm4py.algo.conformance.antialignments.variants.discounted_a_star
↪  import apply as antii
from pm4py.algo.conformance.antialignments.variants.discounted_a_star
↪  import Parameters
from pm4py.evaluation.replay_fitness import evaluator as fitness_eval

if __name__ == '__main__':

    log = xes_importer.apply("2020_dd.xes")
```

```
    log._list=log._list[:1000]
    for threshold in [0.1,0.2,0.5,0.8, 0.9]:
        net, marking, fmarking = inductive_miner.apply(log, parameters =
        ↪  {inductive_minerp.NOISE_THRESHOLD : threshold})
        alignment = ali.apply(log._list[0], net, marking, fmarking)
        clustering = samplingVariantsForAmstc(net, marking, fmarking, log,
        ↪  5, len(alignment)+5 , 2, len(alignment)+5, 2, maxCounter=5,
        ↪  editDistance=True, silent_label="tau", debug=None)
        print("T"+str(threshold),
        ↪  len(clustering)-1,sum([len(clustering[i][1]) for i in
        ↪  range(len(clustering)-1)]), len(clustering[-1][1]))

        resAnti = antii(log, net, marking, fmarking,  parameters =
        ↪  {Parameters.EXPONENT : 1.5,  Parameters.EPSILON : 0.1,
        ↪  Parameters.MARKING_LIMIT : 5})

        fitness = fitness_eval.apply(log, net, marking, fmarking,
        ↪  variant=fitness_eval.ALIGNMENT_BASED)
        print("Fitness", fitness, "Precision:", resAnti['precision'])

    for threshold in [0.9,0.99,0.999,0.9999]:
        net, marking, fmarking = heuristic_miner.apply(log,parameters =
        ↪  {heuristic_miner.Variants.CLASSIC.value.
        ↪  Parameters.DEPENDENCY_THRESH:threshold})
        alignment = ali.apply(log._list[0], net, marking, fmarking)
        clustering = samplingVariantsForAmstc(net, marking, fmarking,
        ↪  log,5, len(alignment)+5 , 2, (len(alignment)+5), 2
        ↪  ,maxCounter=5,editDistance=True,silent_label="tau", debug=None)
        print("T"+str(threshold),
        ↪  len(clustering)-1,sum([len(clustering[i][1]) for i in
        ↪  range(len(clustering)-1)]), len(clustering[-1][1]))

        resAnti = antii(log,net,marking,fmarking,
        ↪  parameters={Parameters.EXPONENT:1.2, Parameters.EPSILON:0.1,
        ↪  Parameters.MARKING_LIMIT:5})
        fitness = fitness_eval.apply(log,net, marking, fmarking,
        ↪  variant=fitness_eval.ALIGNMENT_BASED)
        print("Fitness",fitness,"Precision:",resAnti['precision'])
```

Bibliography

[1] Idc's global datasphere forecast shows continued steady growth in the creation and consumption of data. https://www.idc.com/getdoc.jsp?containerId=prUS46286020. 08 May 2020.

[2] Linkedin's 2020 u.s. emerging jobs report. https://business.linkedin.com/content/dam/me/business/en-us/talent-solutions/emerging-jobs-report/Emerging_Jobs_Report_U.S._FINAL.pdf.

[3] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[4] Arya Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Department of Mathematics and Computer Science, 2014.

[5] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Measuring precision of modeled behavior. *Inf. Syst. E-Business Management*, 13(1):37–67, 2015.

[6] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F van Dongen, and Wil MP van der Aalst. Alignment based precision checking. In *International Conference on Business Process Management*, pages 137–149. Springer, 2012.

[7] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. Memory-efficient alignment of observed and modeled behavior. *BPM Center Report*, 3:1–44, 2013.

[8] 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.

[9] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated

discovery of process models from event logs: Review and benchmark. *IEEE transactions on knowledge and data engineering*, 31(4):686–705, 2018.

[10] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. Split miner: automated discovery of accurate and simple business process models from event logs. *Knowledge and Information Systems*, 59(2):251–284, 2019.

[11] Nithish Pai Ballambettu, Mahima Agumbe Suresh, and R. P. Jagadeesh Chandra Bose. Analyzing process variants to understand differences in key performance indices. In *AISE*, pages 298–313. Springer, 2017.

[12] Martin Bauer, Arik Senderovich, Avigdor Gal, Lars Grunske, and Matthias Weidlich. How much event data is enough? a statistical framework for process discovery. In *International Conference on Advanced Information Systems Engineering*, pages 239–256. Springer, 2018.

[13] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[14] Alessandro Berti and Wil MP van der Aalst. Reviving token-based replay: Increasing speed while improving diagnostics. In *ATAED@ Petri Nets/ACSD*, pages 87–103, 2019.

[15] 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.

[16] Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. A visual approach to spot statistically-significant differences in event logs based on process metrics. In *AISE*. Springer, 2016.

[17] Alfredo Bolt, Massimiliano de Leoni, and Wil M.P. van der Aalst. Process variant comparison: Using event logs to detect differences in behavior and business rules. *Information Systems*, 74:53–66, 2018.

[18] Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. Encoding conformance checking artefacts in sat. In *International Conference on Business Process Management*, pages 160–171. Springer, 2019.

[19] Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. Generalized alignment-based trace clustering of process behavior. In *Proceedings of the 40th International Conference on Applications and Theory of Petri Nets (ICATPN'19)*, number 11522 in Lecture Notes in Computer Science. Springer, 2019.

[20] Mathilde Boltenhagen, Benjamin Chetioui, and Laurine Huber. An alignment cost-based classification of log traces using machine-learning. In *First International Workshop on Leveraging Machine Learning in Process Mining-ML4PM2020*, 2020.

[21] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *Business Process Management Workshops, BPM 2009 International Workshops, Revised Papers*, pages 170–181, 2009.

[22] RP Jagadeesh Chandra Bose and Wil MP Van der Aalst. Context aware trace clustering: Towards improving process mining results. In *proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412. SIAM, 2009.

[23] J.C.A.M. Buijs. Loan application example. 4TU. Centre for Research Data. Dataset. doi.org/10.4121, 2013.

[24] Igor Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Model-based clustering and visualization of navigation patterns on a web site. *Data mining and knowledge discovery*, 7(4):399–424, 2003.

[25] Josep Carmona, Boudewijn van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance checking*. Springer, 2018.

[26] Humberto Carrillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM journal on applied mathematics*, 48(5):1073–1082, 1988.

[27] Tullio Ceccherini-Silberstein, Machı Antonio, and Fabio Scarabotti. On the entropy of regular languages. *Theoretical computer science*, 307(1):93–102, 2003.

[28] Abel Armas Cervantes, Nick RTP van Beest, Marcello La Rosa, Marlon Dumas, and Luciano García-Bañuelos. Interactive and incremental business process model repair. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 53–74. Springer, 2017.

[29] SC Chan, Andrew KC Wong, and David KY Chiu. A survey of multiple sequence comparison methods. *Bulletin of mathematical biology*, 54(4):563–598, 1992.

[30] Thomas Chatain, Mathilde Boltenhagen, and Josep Carmona. Anti-alignments—measuring the precision of process models and event logs. *Information Systems*, page 101708, 2020.

[31] Thomas Chatain and Josep Carmona. Anti-alignments in conformance checking–the dark side of process models. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 240–258. Springer, 2016.

[32] Thomas Chatain, Josep Carmona, and Boudewijn Van Dongen. Alignment-based trace clustering. In *International Conference on Conceptual Modeling*, pages 295–308. Springer, 2017.

[33] Thomas Chatain, Josep Carmona, and Boudewijn F. van Dongen. Alignment-based trace clustering. In *Conceptual Modeling - 36th International Conference, ER 2017, Proceedings*, pages 295–308, 2017.

[34] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1-2):117–136, 1995.

[35] Raffaele Conforti, Marcello La Rosa, and Arthur HM ter Hofstede. Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):300–314, 2016.

[36] Carsten Cordes, Thomas Vogelgesang, and Hans-Jürgen Appelrath. A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In *BPM Workshops*, pages 383–394. Springer, 2015.

[37] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 24–33, 2019.

[38] Ian Davidson, SS Ravi, and Leonid Shamis. A sat-based framework for efficient constrained clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105. SIAM, 2010.

[39] Pieter De Koninck and Jochen De Weerdt. Scalable mixed-paradigm trace clustering using super-instances. In *2019 International Conference on Process Mining (ICPM)*, pages 17–24. IEEE, 2019.

[40] Pieter De Koninck, Jochen De Weerdt, and Seppe KLM vanden Broucke. Explaining clusterings of process instances. *Data mining and knowledge discovery*, 31(3):774–808, 2017.

[41] Pieter De Koninck, Klaas Nelissen, Bart Baesens, Seppe vanden Broucke, Monique Snoeck, and Jochen De Weerdt. An approach for incorporating expert knowledge in trace clustering. In *International Conference on Advanced Information Systems Engineering*, pages 561–576. Springer, 2017.

[42] Pieter De Koninck, Klaas Nelissen, Seppe Vanden Broucke, Bart Baesens, Monique Snoeck, and Jochen De Weerdt. Expert-driven trace clustering with instance-level constraints. *Knowledge and Information Systems*, 63(5):1197–1220, 2021.

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

[44] Massimiliano De Leoni and Wil MP van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1454–1461, 2013.

[45] Javier de San Pedro and Jordi Cortadella. Mining structured Petri nets for the visualization of process behavior. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 839–846, 2016.

[46] Jochen De Weerdt and Seppe vanden Broucke. Secpi: searching for explanations for clustered process instances. In *International Conference on Business Process Management*, pages 408–415. Springer, 2014.

[47] Pavlos Delias, Michael Doumpos, Evangelos Grigoroudis, Panagiotis Manolitzas, and Nikolaos Matsatsinis. Supporting healthcare management decisions via robust clustering of event logs. *Knowledge-Based Systems*, 84:203–213, 2015.

[48] William Edwards Deming and Deming W Edwards. *Quality, productivity, and competitive position*, volume 183. Massachusetts Institute of Technology, Center for advanced engineering study …, 1982.

[49] Marlon Dumas, Wil van der Aalst, and Arthur Ter Hofstede. *Process aware information systems*, volume 1. Wiley Online Library, 2005.

[50] Joseph C Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 1973.

[51] Sean R Eddy et al. Multiple alignment using hidden markov models. In *Ismb*, volume 3, pages 114–120, 1995.

[52] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[53] Javier Esparza. Decidability and complexity of petri net problems—an introduction. In *Advanced Course on Petri Nets*, pages 374–428. Springer, 1996.

[54] Javier Esparza and Mogens Nielsen. Decidability issues for petri nets. *Petri nets newsletter*, 94:5–23, 1994.

[55] Joerg Evermann, Tom Thaler, and Peter Fettke. Clustering traces using sequence alignment. In *International Conference on Business Process Management*, pages 179–190. Springer, 2016.

[56] Dirk Fahland and Wil MP van der Aalst. Repairing process models to reflect reality. In *International conference on business process management*, pages 229–245. Springer, 2012.

[57] Mohammadreza Fani Sani, Mathilde Boltenhagen, and Wil van der Aalst. Prototype selection based on clustering and conformance metrics for model discovery. *arXiv*, 2019.

[58] Diogo R. Ferreira, Marielba Zacarias, Miguel Malheiros, and Pedro Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In *Business Process Management, 5th International Conference, BPM 2007, Proceedings*, pages 360–374, 2007.

[59] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[60] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[61] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 48(2):429–463, 2016.

[62] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[63] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.

[64] Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

[65] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[66] Anders Haug, Frederik Zachariassen, and Dennis Van Liempd. The costs of poor data quality. *Journal of Industrial Engineering and Management (JIEM)*, 4(2):168–193, 2011.

[67] Bart Hompes, Joos Buijs, Wil van der Aalst, Prabhakar Dixit, and Hans Buurman. Discovering deviating cases and process variants using trace clustering. In *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC 2015)*, 2015.

[68] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Pysat: A python toolkit for prototyping with sat oracles. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 428–437. Springer, 2018.

[69] Neil D Jones, Lawrence H Landweber, and Y Edmund Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4(3):277–299, 1977.

[70] Anna Kalenkova and Artem Polyvyanyy. A spectrum of entropy-based precision and recall measurements between partially matching designed and observed processes. In *International Conference on Service-Oriented Computing*, pages 337–354. Springer, 2020.

[71] Marcello La Rosa, Hajo A Reijers, Wil MP Van Der Aalst, Remco M Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. Apromore: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029–7040, 2011.

[72] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

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

[74] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery and conformance checking. *Softw. Syst. Model.*, 17(2):599–631, 2018.

[75] Sander J. J. Leemans, Dirk Fahland, and Wil MP van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013.

[76] X Lu. Using behavioral context in process mining: exploration, preprocessing and analysis of event data. 2018.

[77] Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. Conformance checking based on partially ordered event data. In *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, Revised Papers*, pages 75–88, 2014.

[78] Xixi Lu, Seyed Amin Tabatabaei, Mark Hoogendoorn, and Hajo A Reijers. Trace clustering on very large event data in healthcare using frequent sequence patterns. In *International Conference on Business Process Management*, pages 198–215. Springer, 2019.

[79] Daniela Luengo and Marcos Sepúlveda. Applying clustering in process mining to find different versions of a business process that changes over time. In *International Conference on Business Process Management*, pages 153–158. Springer, 2011.

[80] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[81] Lisa Luise Mannel and Wil MP van der Aalst. Finding uniwired petri nets using estminer. In *International Conference on Business Process Management*, pages 224–237. Springer, 2019.

[82] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained clustering using sat. In *International Symposium on Intelligent Data Analysis*, pages 207–218. Springer, 2012.

[83] Andrey Mokhov, Jordi Cortadella, and Alessandro de Gennaro. Process windows. In *17th International Conference on Application of Concurrency to System Design, ACSD 2017*, pages 86–95, 2017.

[84] Makoena Moloto, Anneke Harmse, and Tranos Zuva. Impact of agile methodology use on project success in organizations-a systematic literature review. In *Proceedings of the Computational Methods in Systems and Software*, pages 267–280. Springer, 2020.

[85] 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.

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

[87] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.

[88] Hoang Huy Nguyen, Marlon Dumas, Marcello La Rosa, and Arthur H.M. ter Hofstede. Multi-perspective comparison of business process variants based on event logs (extended paper). April 2018.

[89] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[90] A. Pini, R. Brown, and M. T. Wynn. Process visualization techniques for multi-perspective process comparisons. pages 183–197. Springer, 2015.

[91] 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. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(3):1–41, 2020.

[92] Hernán Ponce de León, César Rodríguez, and Josep Carmona. POD - A tool for process discovery using partial orders and independence information. In *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015)*, pages 100–104, 2015.

[93] Hernán Ponce de León, César Rodríguez, Josep Carmona, Keijo Heljanko, and Stefan Haar. Unfolding-based process discovery. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Proceedings*, pages 31–47. Springer, 2015.

[94] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):1–36, 2018.

[95] Steven Prestwich, Des Higgins, and Orla O'Sullivan. A sat-based approach to multiple sequence alignment. In *International Conference on Principles and Practice of Constraint Programming*, pages 940–944. Springer, 2003.

[96] Steven David Prestwich. Cnf encodings. *Handbook of satisfiability*, 185:75–97, 2009.

[97] Gang Qian, Shamik Sural, Yuelong Gu, and Sakti Pramanik. Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1232–1237, 2004.

[98] Thomas C Redman. Bad data costs the us $3 trillion per year. *Harvard Business Review*, 22:11–18, 2016.

[99] Hajo A Reijers and S Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005.

[100] Lars Reinkemeyer. *Process Mining in Action*. Springer, 2020.

[101] Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Abel Armas-Cervantes. Scalable conformance checking of business processes. In *OTM CoopIS, , Rhodes, Greece*, pages 607–627, 2017.

[102] Daniel Reißner, Abel Armas-Cervantes, Raffaele Conforti, Marlon Dumas, Dirk Fahland, and Marcello La Rosa. Scalable alignment of process models and event logs: An approach based on automata and s-components. *Information Systems*, 94:101561, 2020.

[103] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[104] Anne Rozinat and Wil MP Van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[105] Vladimir Rubin, Irina Lomazova, and Wil MP van der Aalst. Agile development with software process mining. In *Proceedings of the 2014 international conference on software and system process*, pages 70–74, 2014.

[106] Nikita Saxena, Priyanka Gupta, Ruchir Raman, and Anurag S Rathore. Role of data science in managing covid-19 pandemic. *Indian Chemical Engineer*, pages 1–11, 2020.

[107] Walter Andrew Shewhart and William Edwards Deming. *Statistical method from the viewpoint of quality control*. Courier Corporation, 1986.

[108] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, pages 109–120, 2008.

[109] 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.

[110] Niek Tax, Natalia Sidorova, Reinder Haakma, and Wil MP van der Aalst. Mining local process models. *Journal of Innovation in Digital Ecosystems*, 3(2):183–196, 2016.

[111] Farbod Taymouri and Josep Carmona. A recursive paradigm for aligning observed behavior of large structured process models. In *14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, September 18 - 22*, 2016.

[112] Farbod Taymouri and Josep Carmona. Model and event log reductions to boost the computation of alignments. In Paolo Ceravolo, Christian Guetl, and Stefanie Rinderle-Ma, editors, *Data-Driven Process Discovery and Analysis*, pages 1–21. Springer International Publishing, 2018.

[113] Farbod Taymouri and Josep Carmona. Computing alignments of well-formed process models using local search. *ACM Trans. Softw. Eng. Methodol.*, 29(3):15:1–15:41, 2020.

[114] Farbod Taymouri, Marcello La Rosa, Marlon Dumas, and Fabrizio Maria Maggi. Business process variant analysis: Survey and classification. *Knowledge-Based Systems*, 211:106557, 2021.

[115] Farbod Taymouri, Marcello La Rosa, Marlon Dumas, and Fabrizio Maria Maggi. Business process variant analysis: Survey and classification, 2019.

[116] Nick R. T. P. van Beest, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Log delta analysis: Interpretable differencing of business process event logs. In *BPM*. Springer, 2015.

[117] Wil Van Der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 3(2):1–17, 2012.

[118] Wil Van Der Aalst. Data science in action. In *Process mining*. Springer, 2016.

[119] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011.

[120] Wil Van Der Aalst, Joos Buijs, and Boudewijn Van Dongen. Towards improving the representational bias of process mining. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 39–54. Springer, 2011.

[121] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering*, 16(9):1128–1142, 2004.

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

[123] Wil MP van der Aalst. Re-engineering knock-out processes. *Decision Support Systems*, 30(4):451–468, 2001.

[124] Wil MP Van der Aalst. Using process mining to bridge the gap between bi and bpm. *IEEE Computer*, 44(12):77–80, 2011.

[125] Jan Martijn EM Van der Werf, Boudewijn F van Dongen, Cor AJ Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *International conference on applications and theory of petri nets*, pages 368–387. Springer, 2008.

[126] Boudewijn Van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. Aligning modeled and observed behavior: A compromise between computation complexity and quality. In *International Conference on Advanced Information Systems Engineering*, pages 94–109. Springer, 2017.

[127] Boudewijn F van Dongen. Bpi challenge 2015. In *11th International Workshop on Business Process Intelligence (BPI 2015)*, 2015.

[128] Boudewijn F van Dongen. Efficiently computing alignments: using the extended marking equation. In *16th International Conference on Business Process Management, BPM 2018*, pages 197–214. Springer, 2018.

[129] Boudewijn F van Dongen, Josep Carmona, and Thomas Chatain. A unified approach for measuring precision and generalization based on anti-alignments. In *International Conference on Business Process Management*, pages 39–56. Springer, 2016.

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

[131] 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. *Int. J. Data Sci. Anal.*, 8(3):269–284, 2019.

[132] Sebastiaan J van Zelst, Alfredo Bolt, and Boudewijn F van Dongen. Tuning alignment computation: An experimental evaluation. In *ATAED@ Petri Nets/ACSD*, pages 6–20, 2017.

[133] Sebastiaan J van Zelst and Yukun Cao. A generic framework for attribute-driven hierarchical trace clustering. In *International Conference on Business Process Management*, pages 308–320. Springer, 2020.

[134] 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.

[135] Seppe KLM vanden Broucke, Jorge Munoz-Gama, Josep Carmona, Bart Baesens, and Jan Vanthienen. Event-based real-time decomposed conformance analysis. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 345–363. Springer, 2014.

[136] H. M. W. Verbeek and W. M. P. van der Aalst. *Merging Alignments for Decomposed Replay*, pages 219–239. Springer International Publishing, Cham, 2016.

[137] Silke Wagner and Dorothea Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[138] Jochen De Weerdt, Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25(12):2708–2720, 2013.

[139] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[140] Lijie Wen, Wil MP Van Der Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

[141] Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[142] Glynn Winskel. Petri nets, algebras, morphisms, and compositionality. *Information and Computation*, 72(3):197–238, 1987.

[143] M.T. Wynn, E. Poppe, J. Xu, A.H.M. ter Hofstede, R. Brown, A. Pini, and W.M.P. van der Aalst. Processprofiler3d: A visualisation framework for log-based process performance comparison. *DSS*, 100, 2017.

[144] Fareed Zandkarimi, Jana-Rebecca Rehse, Pouya Soudmand, and Hartmut Hoehle. A generic framework for trace clustering in process mining. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 177–184. IEEE, 2020.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Partitionnement d'instances de processus basé sur les techniques de conformité de modèles

Mots clés: Process Mining, Vérification de Conformité, Partitionnement

Résumé: Les données d'événements devenant une source d'information omniprésente, les techniques d'analyse de données représentent une opportunité sans précédent pour étudier et réagir aux processus qui génèrent ces données. Le Process Mining est un domaine émergent qui comble le fossé entre les techniques d'analyse de données, comme le Data Mining, et les techniques de management des entreprises, à savoir, le Business Process Management. L'une des bases fondamentales du Process Mining est la découverte de modèles de processus formels tels que les réseaux de Petri ou les modèles BPMN qui tentent de donner un sens aux événements enregistrés dans les journaux. En raison de la complexité des données d'événements, les algorithmes de découverte de processus ont tendance à créer des modèles de processus denses, qui sont difficiles à interpréter par les humains. Heureusement, la Vérification de Conformité, un sous-domaine du Process Mining, permet d'établir des liens entre le comportement observé et le comportement modélisé, facilitant ainsi la compréhension des correspondance entre ces deux éléments d'information sur les processus. La Vérification de Conformité est possible grâce aux artefacts d'alignement, qui associent les modèles de processus et les journaux d'événements. Il existe différents types d'artefacts d'alignement, à savoir les alignements, les multi-alignements et les anti-alignements. Actuellement, seuls les alignements sont traités en profondeur dans la littérature scientifique. Un alignement permet de relier le modèle de processus à une instance de processus donnée. Cependant, étant donné que de nombreux comportements existent dans les logs, l'identification d'un alignement par instance de processus nuit à la lisibilité des relations log-modèle. La présente thèse propose d'exploiter les artefacts de conformité pour regrouper les exécutions de processus enregistrées dans les journaux d'événements, et ainsi extraire un nombre restrictif de représentations modélisées. Le regroupement de données, communément appelé partitionnement, est une méthode courante pour extraire l'information de données denses et complexes. En regroupant les objets par similarité dans des clusters, le partitionnement permet d'extraire des ensembles de données plus simples qui englobent les similarités et les différences contenues dans les données. L'utilisation des artefacts de conformité dans une approche de partitionnement permet de considérer un modèle de processus fiable comme une base de référence pour le regroupement des instances de processus. Ainsi, les clusters découverts sont associés à des artefacts modélisés, que nous appelons variantes modélisées des traces, ce qui fournit des explications opportunes sur les relations entre le journal et le modèle. Avec cette motivation, nous avons élaboré un ensemble de méthodes pour calculer les artefacts de conformité. La première contribution est le calcul d'un comportement modélisé unique qui représente un ensemble d'instances de processus, à savoir le multi-alignement. Ensuite, nous proposons plusieurs approches de partitionnement basées sur l'alignement qui fournissent des clusters d'instances de processus associés à un artefact modélisé. Enfin, nous soulignons l'intérêt de l'anti-alignement pour extraire les déviations des modèles de processus par rapport au journal. Ce dernier artefact permet d'estimer la précision du modèle. Nous montrons son impact sur nos approches de partitionnement basées sur des modèles. Nous fournissons un encodage SAT pour toutes les techniques proposées. Des heuristiques sont ensuite ajoutées pour tenir compte de la capacité de calcul des ordinateurs actuels, au prix d'une perte d'optimalité.

Title: Process Instance Clustering Based on Conformance Checking Artefacts

Keywords: Process Mining, Conformance Checking, Clustering

Abstract: As event data becomes an ubiquitous source of information, data science techniques represent an unprecedented opportunity to analyze and react to the processes that generate this data. Process Mining is an emerging field that bridges the gap between traditional data analysis techniques, like Data Mining, and Business Process Management. One core value of Process Mining is the discovery of formal process models like Petri nets or BPMN models which attempt to make sense of the events recorded in logs. Due to the complexity of event data, automated process discovery algorithms tend to create dense process models which are hard to interpret by humans. Fortunately, Conformance Checking, a sub-field of Process Mining, enables relating observed and modeled behavior, so that humans can map these two pieces of process information. Conformance checking is possible through alignment artefacts, which associate process models and event logs. Different types of alignment artefacts exist, namely alignments, multi-alignments and anti-alignments. Currently, only alignment artefacts are deeply addressed in the literature. It allows to relate the process model to a given process instance. However, because many behaviors exist in logs, identifying an alignment per process instance hinders the readability of the log-to-model relationships. The present thesis proposes to exploit the conformance checking artefacts for clustering the process executions recorded in event logs, thereby extracting a restrictive number of modeled representatives. Data clustering is a common method for extracting information from dense and complex data. By grouping objects by similarities into clusters, data clustering enables to mine simpler datasets which embrace the similarities and the differences contained in data. Using the conformance checking artefacts in a clustering approach allows to consider a reliable process model as a baseline for grouping the process instances. Hence, the discovered clusters are associated with modeled artefacts, that we call model-based trace variants, which provides opportune log-to-model explanations. From this motivation, we have elaborated a set of methods for computing conformance checking artefacts. The first contribution is the computation of a unique modeled behavior that represents of a set of process instances, namely multi-alignment. Then, we propose several alignment-based clustering approaches which provide clusters of process instances associated to a modeled artefact. Finally, we highlight the interest of anti-alignment for extracting deviations of process models with respect to the log. This latter artefact enables to estimate model precision, and we show its impact in model-based clustering. We provide SAT encoding for all the proposed techniques. Heuristic algorithms are then added to deal with computing capacity of today's computers, at the expense of loosing optimality.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France