Light Methods for the Conformance Checking of Business Processes

Doctorate of Philosophy Dissertations

by

Farbod Taymouri
Supervisor:
Prof. Josep Carmona

School of Computer Science
Polytechnic University of Catalonia (Barcelona Tech)

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

November 9, 2018

Keywords: PhD Dissertation, UPC, Data Science, Process Mining, Computing.

In accordance with the requirements of the degree of Doctor of Philosophy in Computer Science in the School of Computer Science, I present the following thesis entitled,

Light Methods for the Conformance Checking of Business Processes
Doctorate of Philosophy Dissertations

This work was performed under the supervision of Professor Josep Carmona. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at Polytechnic University of Catalonia (Barcelona Tech) or any other institution.

Farbod Taymouri

To those who seek knowledge for the satisfaction of the mind,
and understanding of the world

Acknowledgements

First and foremost, I want to express my deepest gratitude and thanks to my advisor Prof. Josep Carmona. It has been an honor to be his Ph.D. student. He has taught me, both consciously and unconsciously, how a good research is done, and how to be a researcher. I appreciate all his contributions of time, ideas, comments and supporting to make my Ph.D. degree. The joy and enthusiasm he has for his research was contagious and motivational for me, even during tough times in the Ph.D. pursuit.

Also, I would like to express my sincere thanks and appreciation to Prof. Boudewijn van Dongen, chair of the Process Analytics group at Eindhoven University of Technology (TU/e). Your comments and points have had significant impacts on my research.

I would also like to thank my committee members for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

A special thanks to my family. Words can not express how grateful I am to my father, mother, and my sister for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far.

Abstract

In Process Mining, Conformance Checking of business process models addresses the deviations happened in their real executions. Though it is a very important matter on its own by detecting where and which deviations happened, more importantly it is a vital issue for organizations and many other process mining techniques as well. For example, in model repair techniques it opens a door for enhancement of processes. Indeed, identifying these deviations boils down to the notion of alignment conceptually. An alignment quantifies to what degree a process model can imitate what happened in its observed behavior, i.e., an event log. In fact, an alignment is a matrix data structure with two rows, where each column represents whether an observed event can be lined up with the corresponding element of the given process model. With that said, an optimal alignment is the best combination by which the process model can imitate the corresponding observed behavior. State of the art technique for alignment computation has exponential time and space complexity, hindering its applicability for medium and large instances.

The main aim of this thesis is to propose light and efficient methods for alignment computation. Above that, the second aim is to tackle the mentioned challenge in a big picture point of view, namely not considering all details. In this case, the focus will be on fatal deviations. Doing this not only alleviates the computational challenge, but also provides a greater logical perspective of deviations.

All methods presented in this thesis are forms of combinatorial optimizations for alignment computation. They make close approximation to an optimal solution. The corresponding contributions represent novel ways in which they are proposed and their effectivenesses for the mentioned challenge with respect to state of the art approach in different perspectives. Generally speaking, methods presented in this thesis can be categorized as:

- **Classical approaches:** These techniques exploit Integer Linear Programming (ILP) as well as structural theory of Petri nets to formulate alignment computation as an optimization of a set of linear equations. A modification to this strategy which trades-off between complexity and quality is to integrate it with state of the art approach.

- **Heuristic approaches:** These techniques adopt different policy with respect to the first approach, and take advantages of heuris-

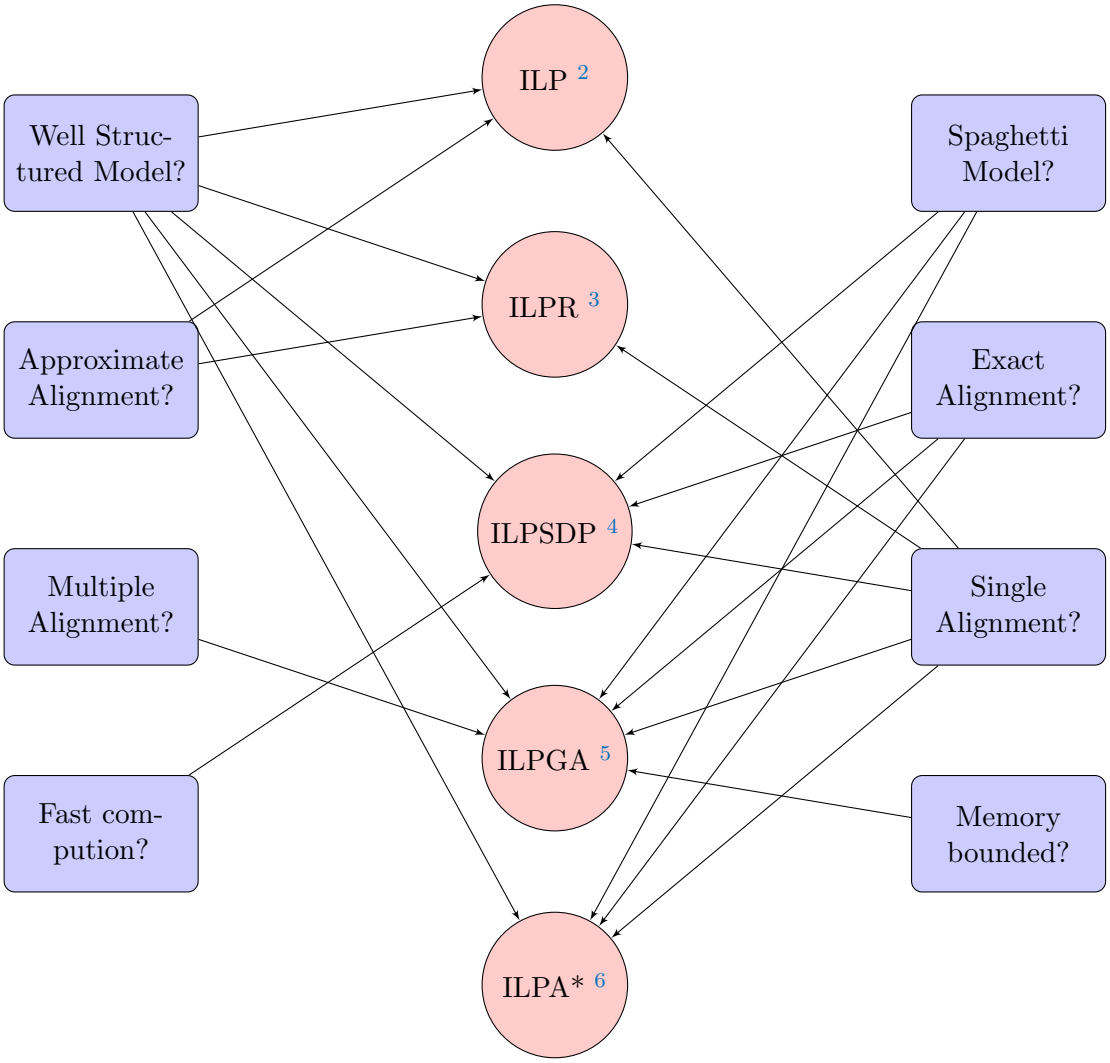
tic functions to explore the search space of alignments to find the optimal one(s). This can be done by obtaining an initial solution and iteratively improving it until saturation or reaching a certain criterion. Another way could be by adopting a Genetic Algorithm with well specific designed operators, by which exploration of the corresponding search space can be speed up toward the best solution(s).

- **Model reduction:** Regardless of the two main approaches just mentioned, one way to boost the effectiveness of alignment computation is reducing model and observed behavior without loosing alignment information. This structure reduction not only boosts the alignment computation, but also provides a big picture of detected deviations. Above that, a divide-and-conquer strategy will be provided for the first approach such that it breaks the original problem into a set of smaller independent problems which can be solved independently.

Experiments witness the merit of proposed approaches with respect to state of the art in different perspectives such as resource consumptions, execution time, quality and accuracy of solutions found. All methods have been implemented as a stand-alone tool box called ALI [1] [73]. The tool box and datasets used in this thesis are publicly available at https://www.cs.upc.edu/~taymouri/.

[1] Alignment of Large Instances

The following plot summarizes the presented techniques in this thesis based on different application demands.



[2] Integer Linear Programming (ILP) approach to alignment computation, Chapter 3.
[3] Recursive Integer Linear Programming (ILPR) approach to alignment computation, Chapter 7.
[4] Integer Linear Programming Sequential Dynamic Programming (ILSDP) approach to alignment computation, Chapter 5.
[5] Integer Linear Programming Genetic Algorithm (ILPGA) approach to alignment computation, Chapter 6.
[6] Integer Linear Programming A* (ILPA*) approach to alignment computation, Chapter 4.

Contents

Part I

Introduction

This part starts with a brief introduction to process mining area and the problem, i.e., alignment computation, which has been tackled in this thesis. Next, the importance of alignment computation as well as motivation is presented. Following that, the contribution of proposed algorithms in this thesis will be given, and finally the required preliminaries and related work are provided.

Chapter 1

Alignment in Conformance Checking

1.1 Introduction

Big Data has gained much attention from the academia, IT and business industry. In the digital and computing world, information is generated and collected at a rate that rapidly exceeds the boundary range. Hence organizations are facing a digital transformation, that primarily requires an active use of the tones of data available as a result of their operation. Big Data can be described in four dimensions as [38]:

- *Volume*: Huge amount of data can provide better understanding and more comprehensive insights about the business in different aspects. In effect making better decisions is affected by having more confidence about the phenomenon under consideration.

- *Velocity*: The more rapidly information can be made from available data, the more flexibility will be obtained to find answers in terms of questions via queries, reports, dashboards, etc. A rapid data ingestion and rapid analysis capability provides timely and correct decision performances.

- *Variety*: The more heterogeneous data types are available, from the CRM system, social media, call-center logs, etc, the more multi-faceted view can be developed about the business under consideration, thus enabling an analyst to develop more customized businesses.

- *Veracity*: Managing a lot of data does not mean the data becomes clean and accurate. Business data must remain consolidated, cleansed, consistent, and current to make the right decisions.

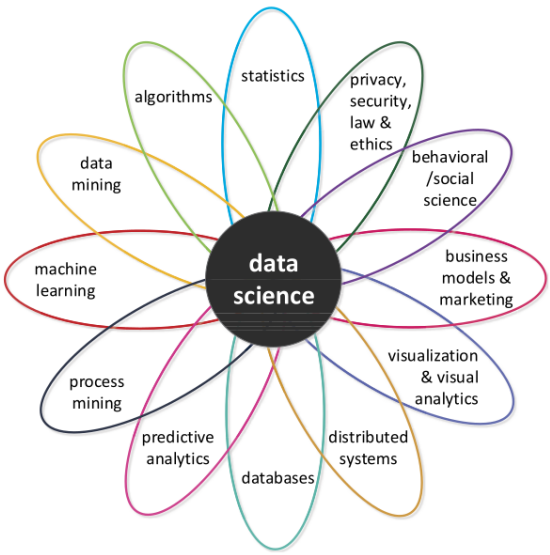


Figure 1.1: Data Science techniques [3]

Despite of living in Big Data age, the naive goal would not be to collect as much data as possible. The real challenge is to make valuable insights from this information. Most of the data stored in the digital universe is unstructured, and therefore organizations have problems dealing with such large quantities of data. One of the main challenges of todays organizations is to extract information and value from data stored in their information systems. The importance of information systems is not only reflected by the massive growth of data, but also by the role that these systems play in todays business processes as the digital universe, and the physical universe are becoming more and more aligned [3]. Indeed this amount of data must be managed efficiently. This has had an incredible impact on the way that businesses are communicated [49]. For example, the state of a retailer which sells many products and goods is mainly determined by the data in its information system and when a purchase is done by a customer, the retailer makes interactions with many organizations like banks, brokers and etc. often without being aware of its state. If a purchase is successful the customer receives purchased item. The activities of a *business process* like the one just mentioned can be recorded which is called *event logs*. Events might take place everywhere inside a machine (e.g., an X-ray machine, an ATM, or baggage handling system), an enterprise information system, a hospital, a social network, and a transportation system.

Classical data analysis methods can not take full advantages of other available resources and data, hence the need for a new discipline of *data-driven science* is needed for this issue. *Data Science* proposes modern and scalable data analysis techniques which tackle the mentioned challenges from different angles, to take fully advantages of available data. Under the Data Science umbrella, *Process Mining* is a set of techniques and algorithms that deal with event logs of processes, see Fig. 1.1. Only process mining techniques directly relate event data to end-to-end business processes [3]. The growing interest is illustrated by the Process Mining Manifesto [83] released by the IEEE Task Force on Process Mining some years ago.

Process mining is a relative young and fast growing research discipline, that sits between traditional analysis paradigm like machine learning and data mining on the one hand, and process modeling and analysis on the other hand. It can be viewed as the link between data science and process science. Process mining seeks how to confront event data, i.e., observed behavior, and process models, i.e, de facto model which can be hand-made or discovered automatically. Data mining, statistics and machine learning techniques do not consider end-to-end process models. Process science approaches are process- centric, but often focus on modeling rather than learning from event data or estimating parameters. The unique positioning of process mining makes it a novel strong tool to exploit the growing availability of data for improving end-to-end processes.

Process mining can be best related to Business Process Management, i.e., BPM, where the main focus of BPM is on process design and imple-

mentation [4], [24]. Process modeling plays a key role in the (re)design phase and directly contributes to the configuration/implementation phase. Originally, BPM approaches had a tendency to be model-driven without considering the evidence hidden in the data. However, process mining is not limited to BPM, and any process for which events can be recorded, is a candidate for process mining [3]. A *process* can be seen as a set of related business activities in an organization, for example it can be delivering of products to customers, click streams of users over the Internet for buying some special products or a an application process inside the RAM of a machine which interacts with other processes and Operating System.

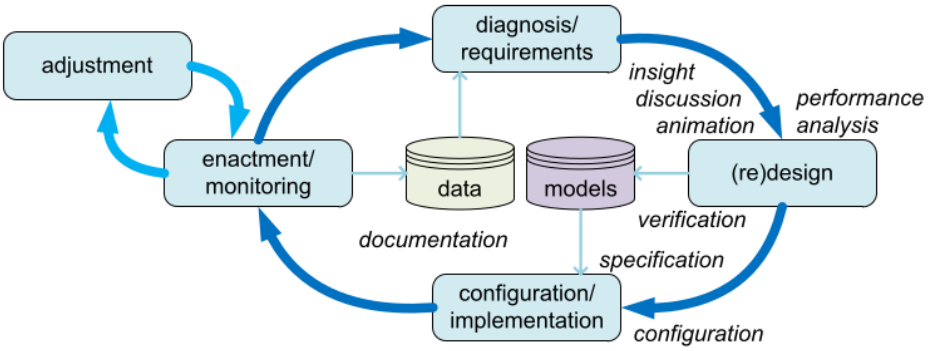


Figure 1.2: Business process management life cycle [3]

The key role of process mining can be described more easily in BPM life-cycle, see Fig. 1.2. It describes the different phases of managing a particular business process. In the *design* phase, a process is designed and running into a system. In *monitoring* phase, the processes are running while being monitored by management to see if any changes are needed. Some of these changes are handled in the *adjustment* phase. In this phase, the process is not redesigned and no new software will be developed only predefined controls are used to adapt or reconfigure the process. The *requirements* phase evaluates the process and monitors emerging requirements due to changes in the environment of the process (e.g., changing policies, laws, competition). Poor performance which results in new requirements triggers the *redesign* phase. Process mining offers the possibility to truly close the BPM life-cycle. Corresponding recorded data can be used to provide a better view on the actual processes, i.e., deviations can be analyzed and the quality of models can be improved.

The general idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in todays systems. In general process mining can be done in three dimensions as follows and shown in Fig. 1.3:

- *Process Discovery*: The goal in this part is to discover a new process model based on provided event logs without any prior information.

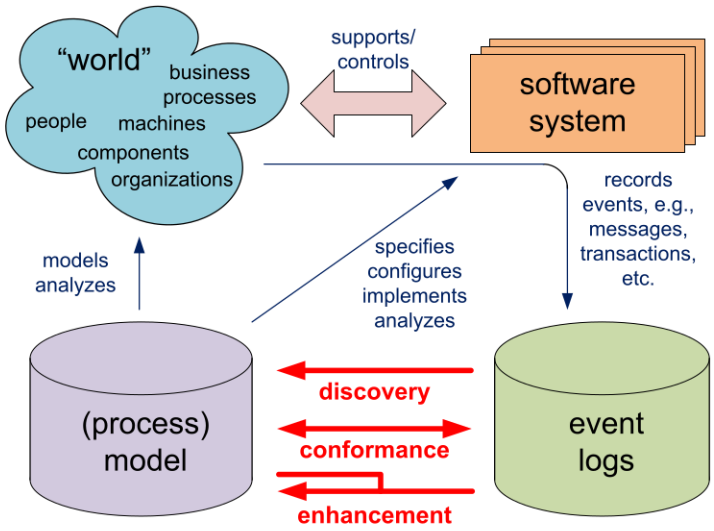


Figure 1.3: Process mining types [3]

Since these algorithms are fed by raw data, the role of these algorithms and the challenges ahead are similar to unsupervised techniques in machine learning for discovering communities in social networks [44]. The first proposed approach for this challenge is called α-algorithm and was presented in [82]. This algorithm has a very restricted representational bias, and hence there are many other approaches which tackle discovering tasks from different perspectives, some of them are as follows, [21], [42], [104], [89].

- *Conformance Checking*: This set of techniques assumes that there are process model and corresponding event logs. The process model can be obtained by hand or by applying a discovery technique. The main objective is to examine whether the footprint agrees with the process model. This can be viewed from different angles which provide useful insights [3],

  1. The model is wrong and unable to reflect the reality.
  2. Cases deviate from the model and thus corrective actions are needed.

  These techniques are very important because they enable auditing and checking whether business processes are executed within certain boundaries set by managers, governments, and other stakeholders. For example, specific rules may be enforced by law or company policies and the auditor should check whether these rules are followed or not [78]. These techniques are similar to supervised learning methods since the existing process model provides information about the

rules that must be followed. Indeed, the existing model is supposed to explain the underlying concept of the given phenomenon.

- *Enhancement*: These set of techniques can be applied from different angles, first and the most important objective aims at improving the existing process model with the use of event logs. That is, the model is repaired or improved if it does not agree with its real execution and some deviations identified. By highlighting the repaired parts of the repaired model, one can show discrepancies succinctly. This may lead to adaptations of the actual process (e.g., better work instructions or more control) or to changes of the model to reflect reality better. In the latter case the repaired model can be used as the new normative or descriptive model [27]. Second, evolving in observed behavior can be monitored and it signals the existing process needs to be updated [28]. The necessity of enhancement or improvement comes from the fact that real world is dynamic and relations are changed over the time. Hence concept drift is expected to happen and thus the corresponding models must be adopted accordingly. These techniques in nature are similar to semisupervised or more specific, active learning techniques, where the learned patterns must be updated over the time.

Process mining has applications in diverse domains. It can be applied in computer science to social science. Conceptually, wherever there is a collection of event logs, one can take advantages of process mining [33]:

- *Usage profiling*: By using process mining one can build models describing the actual use of the application. Such models may reveal that certain features are never used or that they are only used by particular customers under particular circumstances. This helps to characterize the different types of users and may influence further development, training, and marketing [47].

- *Reliability improvement and anomaly detection*: The models discovered through process mining show the actual use of the application and also its failures. Insight into the actual use of the application can be used to test the system under realistic circumstances (sometimes this is even enforced by law). Moreover, a detailed analysis of failures using process mining can help to find root causes for reliability problems [12].

- *Usability improvement.* If the intended use deviates from the actual use, then this may point towards usability problems. Using conformance checking one can locate such deviations and quantify their impact.

- *Remote diagnostics and servicing.* Information in event logs can also be used more actively. For example, process mining can help to

predict failures. By combining current data and historic information, it is possible to forecast likely problems. Moreover, if an error occurs, the event log may be used to find the core problem and take counter measures [3].

Also there are applications in software engineering like [48] where process mining can facilitate evaluation and auditing. Designing social network analysis from email logs with help of process mining is presented in [29] or maintenance of the road and water infrastructure [79]. A massive collections of real and synthetic event logs from different applications can be found in IEEE TF on Process Mining - Event Logs.

This thesis is mainly centered around conformance checking of process models. Before doing so, it would be instructive to shed light on the relation between conformance checking, business alignment, and auditing issues. The goal of business alignment is to make sure that the information systems and the real business processes are well aligned to each other. Unfortunately, there is often a mismatch between the information system on the one hand, and the actual processes and needs of workers and management on the other hand. One of the reasons could be that most organizations use generic software that was not developed for a specific organization. A typical example is the SAP system which is based on so-called best practices, i.e., typical processes and scenarios are implemented [3]. In spite of such systems are configurable, particular needs of an organization may be different from what was developed by the product software developer, that is to say their specific requirements of that organization is not included and indeed it is impossible. Second, processes may change faster than the information system, because of external influences and being in agile environments. Finally, there may be different stakeholders in the organization having conflicting requirements, e.g., a manager may want to enforce a fixed working procedure whereas an experienced worker prefers to have more flexibility to serve customers better.

Conformance checking can assist in improving the confrontation or alignment of information systems, business processes, and the organization. Given a process model and its real execution, by analyzing the later with respect to the former and diagnosing discrepancies between them, new insights can be gathered showing how to improve the support by information systems. Conformance checking between a model and event logs can be viewed and done in four separate dimensions as follows [6], [3]:

- *Fitness*: It measures the proportion of behavior in the observed trace, i.e. log, according to the normative model. More specific how well the model is enable to mimic the observed trace.

- *Precision*: It measures the behavior which is unlikely given the observed races of the event logs. Stated differently, the model is able to behave much beyond of the seen event logs.

- *Generalization*: The discovered model should generalize the example behavior seen in the event log. This dimension examines the discovered model in terms of overfitting.

- *Simplicity*: It measures how much the model is simple and understandable in which it is able to explain the behavior, [50] presents many metrics to quantify the complexity and understandability of a process model, the metrics consider perspectives like size of the model and "structure" of the model.

Abundance of efforts and papers were published recently to show the importance of fitness dimension regarding conformance checking issue.

The backbone of most conformance checking algorithms is the notion of *alignment* introduced in [5], where it denotes which events of the event log can be aligned with the corresponding elements of the process model, and this number must be maximized. This is a challenging combinatorial optimization problem. Different algorithms try to optimize various objective functions [43], [20] and [66]. The complexity of solving these optimization problems increase exponentially to the size of the problem and as a result they are unable to compute alignments for medium to large datasets. In spite of the importance of alignment computation and the mentioned challenge, the aim of this research is to present light methods for the alignment computation i.e., light and efficient computational method of combinatorial optimization. This can be approached based on both classical and heuristics optimization techniques where the corresponded algorithms are presented in parts II, III respectively and following those, methods for alleviating the burden of alignment computation in scale will be presented.

1.2 Motivation

As mentioned in the previous section this thesis is centered around proposing light methods of conformance checking and will be focused on alignment computation. In general the conformance checking of a process model and an event log aims to answer this question: is the real execution, i.e., footprints, of a process model is in accord with its model? If no, how to quantify this situation and more importantly what the deviations are exactly.

This sort of analysis plays a vital role in both process mining and process management, since obtained information not only can provide more insights about the real execution of the process but also this information can be used to improve or repair the existing process model from different perspectives. Above that, this information can be used in usage profiling recommendation and search engine information systems to make a decision about the similarity of an event log to existing process models.

In order to make conformance checking more concrete in terms of the mentioned question, the notion of *alignment* is proposed by [5] which will be formally introduced in preliminaries chapter 2. In short, an alignment

between a process model and an observed trace of event log is a two rows matrix that denotes which elements of the process model can be aligned to each events of the observed trace. To give an idea of what is an alignment lets assume there is a process model like the one shown in Fig. 1.4 in Petri-net formalism with four transitions t_1, t_2, t_3 and t_4 with the corresponding labels.

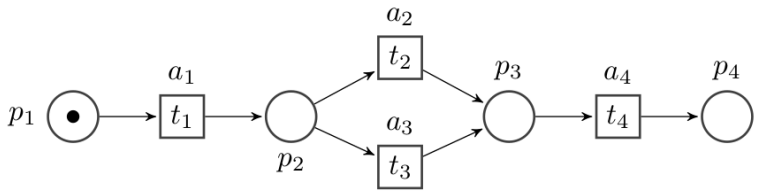


Figure 1.4: Process Model

Also there is an observed behavior $\sigma = a_1 a_1 a_4 a_2$, then one possible alignment between the process model and observed trace could be:

$$\alpha = \begin{array}{|c|c|c|c|c|} \hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline t_1 & \bot & t_3 & t_4 & \bot \\ \hline \end{array}$$

Where each column represents a *move*. Obviously on the one hand if there is no deviation, i.e., footprint is 100% in accord with the process model, then all events can be aligned synchronously with elements of the process model, i.e., *synchronous* move, for example $(a_1, t_1), (a_4, t_4)$ are synchronous moves. On the other hand if there are deviations, some elements of the observed trace can not be aligned synchronously with elements of the process model, i.e., *asynchronous* move, for example $(a_1, \bot), (\bot, t_3)$ are asynchronous moves. In short an alignment can be quantified by assigning a number to it which is called *fitness*, a number between 0 and 1, where a value from this interval represents how well the model can mimic the observed trace.

It is clear that given a process model and an observed trace, by assigning a weight to each move (an asynchronous move receives greater cost than a synchronous move), an alignment with the minimum cost is a desired one, since it shows the maximum similarity between the observed trace and the process model. Stated differently, obtaining a desired alignment just mentioned boils down to a *Combinatorial Optimization Problem*. State of the art approach for alignment computation as presented in [5] is based on A^* algorithm. This algorithm is an informed search method and an extension of *best-first* search methods, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the best cost [51]. In this problem it is defined as the search for a minimal path on the product of the state space of the process model and the observed behavior, an object that is worst-case exponential with respect to the size of the model. This hampers the application of these techniques for medium/large instances.

The efficiency of A^* algorithm depends on the adopted heuristics to prune the respective search space, and in case of having an admissible heuristics it is guaranteed to find optimal solutions. Despite of having an admissible heuristics, the corresponding search space even for medium size of real world problems could be large and makes this kind of search for obtaining an optimal alignment impractical. This issue becomes more challenging when more than one solution is needed.

As mentioned shortly in the previous paragraph, alignment computation is not an easy task, and at the same time too expensive since in technical terms it has exponential complexity to size of a given problem. This research is manly focused on proposing light methods for alignment computation to tackle mentioned challenges, and fulfill the following issues:

- *Scalable methods of alignment computation*: State of the art approach has limitation in dealing with massive process and large even logs. A way to get around of obtaining an optimal solution is to approximate it, which could be much more affordable in terms of resource consumptions, and therefore can be used in an scalable way and much more faster to provide close results to the desired ones.

- *Approximate alignments*: In some applied areas like in medical or health applications, it is not necessary to have all details of an optimal alignment, and having a big picture could be enough. That is to say this is a generalization to the notion of alignment. The advantages are manifold: conformance checking techniques can be discretized to a desired (time) granularity, e.g., when the ordering of activities in a specific period is not important for the diagnosis. Also, other techniques like *model repair* [28] may be guided to only repair coarse-grain deviating model parts. Finally, in domains where a fine-grained ordering of activities is not needed, approximate alignments can play an important role (e.g., health care [45]).

- *Multiple solutions*: Some applications of process management like model repair are heavily based on alignments. In these applications for a given observed trace having different alignments with the same fitness value could be beneficial, since the model can be repaired from different perspectives. Having a set of solutions, i.e., optimal alignments or alignments close to optimal ones, in an affordable way is necessary for these applications.

- *Abstract perspectives*: Alignment computation can be alleviated by reducing the size of a given problem in terms of model and event log reduction. This policy not only simplifies alignment computation, but also provides various perspectives of the given problem at different levels. This kind of abstraction provides better understanding of various parts of big process models, and reveals the main parts of it.

So, as just demonstrated above in parallel to the main objective of this thesis i.e., light methods of alignment computation, other topics will be addressed to fulfill the mentioned challenges and requirements.

1.3 Contributions

Based on explained motivations in the previous section, in this thesis six approaches are presented to fulfill the mentioned issues. None of techniques covers all explained issues simultaneously, but each of them focuses on a portion of challenges as will be explained in corresponding chapters.

All presented approaches are composed of different steps, but the first step in all of them is based on a novel idea which is formulation of Integer Linear Programming (ILP) and structural theory of Petri nets (will be presented in Chapter 3, Section 5.3.1). Stated differently, in this thesis the problem of alignment computation can be tackled in two main optimization steps. The first step as mentioned is based on pure ILP and the second step adopts separate new strategies which vary and depend on the underlying challenges, for example a new genetic algorithm strategy is used to obtain multiple solutions or a local search approach for scalable and fast alignment computation. Contributions of each approach as well as the corresponding publications are shown in Table 1.1:

	Method	Publication
[75]	ILP and Recursive ILP, Chapters 3, 7	*International Conference on Business Process Management (BPM 2016)*
[74]	Structure Reduction, Chapter 8	*International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016)*-**Best Research Paper Award**
[90]	Hybrid Approach, Chapter 4	*International Conference on Advanced Information Systems Engineering (CAISE 2017)*
[76]	Local Search Approach, Chapter 5	The journal paper was submitted to *ACM Transactions on Software Engineering and Methodology*
[77]	Genetic Algorithm Approach, Chapter 8	*International Conference on Business Process Management (BPM 2018)*

Table 1.1: Publications of proposed approaches

- *Integer Linear Programming (ILP)*: This approach presents a novel optimization formulation based on ILP and structural theory of Petri nets for alignment computation. This approach for the first time introduces approximate alignment computation at different granularity levels, and guarantees its optimality. Due to its complexity, i.e., NP-

	ILP	Recursive ILP	Hypbrid ILP & A^*	Local Search	Genetic Algorithm	A^*(State of the art)
Solution	Single	Single	Single	Single	Single & Multiple	Single & Multiple
Optimality	Not Guaranteed	Not Guaranteed	Not Guaranteed	Not Guaranteed	Not Guaranteed	Guaranteed
Scalability	No	Yes (Very)	Yes (Rather)	Yes (Extremely)	Yes (Very)	No
Exact	No (Opt.)[a]	No	Yes	Yes	Yes	Yes
Memory Footprint	Low (BD[b])	Very Low (BD)	Medium (UBD)	Very Low (BD)	Medium (BD)	Very high (UBD)
Execution Speed	Slow	Fast	Dependable	Super fast	Slow	Slow
Complexity[c]	NP-Complete	NP-Complete[d]	NP-Complete	Polynomial	Polynomial[e]	Exponential

Table 1.2: Proposed approaches

[a] In terms of Parikh vector the solution is optimal.

[b] Bounded

[c] It must be note that for the last two approaches, fourth and fifth columns, theoretical complexities are NP-Complete since they use ILP as well. However in practice due to the small size of ILP instance, other parts of algorithms which are Polynomial play important roles.

[d] The size of ILP instances for this approach is very small and therefore in practice it is very efficient.

[e] The complexity of operators are considered.

Hard, it is neither scalable nor fast for fine level of granularities. This approach will be presented in Chapter 3.

- *Recursive Integer Linear Programming (ILPR)*: The complexity of an ILP optimization is NP-Hard, and therefore solving it for large instances is impossible. Thus the only way to make it scalable is to reduce the size of a problem and, at the same time, to preserve features of the original problem to not affect the final solution. The contribution of this approach is the decomposition a large ILP problem to a set of smaller ILPs and solve them recursively such that the original ILP conditions are preserved. In this way the ILP instances can be solved significantly faster than the original one. In spite of increasing scalability, speed and reducing resource consumptions, doing so increase the risk of having spurious solutions. Chapter 7 presents this approach.

- *Hybrid ILP (ILPA*)*: This approach presents a novel hybrid technique based on state of the art approach [5], i.e., A^* and ILP optimization. Stated differently, it represents a trade-off between complexity and quality. The approach depends on desired quality and complexity, and it computes an alignment in a hybrid way, where giving more weight to A^* provides better solutions in terms of quality, but at the same time sacrifices scalability and vise versa. This approach will be discussed in Chapter 4.

- *Local Search (LS)*: This approach and the next one tackle the optimization problem in different ways, using novel heuristics methods. Optimization with the help of heuristic functions can be efficient if the domain knowledge of the problem is injected to these functions. The novelty of this approach is to obtain an initial solution with relaxed ILP optimization, then improve it with the use of local search iteratively. Despite of not having globally optimal solutions in general, this approach is super light, very fast and extensively scalable. This approach will be explored with more details in Chapter 5.

- *Genetic Algorithm (GA)*: None of the previous methods are able to provide multiple solutions, i.e., optimal or close to optimal alignments for a given problem. This approach adopts genetic algorithm framework, and propose novel domain specific operators to accelerate and guide the algorithm into search space of a given problem to where the desired solutions are located. The advantages of this approach lay on the fact that it provides a bunch of solutions at once. It will be discussed in Chapter 6.

- *Reduction Algorithm*: Apart from approaches mentioned in Table 1.2, a novel framework for the reduction of a process model and event log is presented based on detecting special substructures called *Single Entry, Single Exit (SESE)* inside the given process model. The

main objective of this framework is the alleviation of alignment computation, by reducing the size of a given problem. This reduction framework can be integrated with all techniques of alignment computation. Above that, the reduction of a process model is equivalent to its structural summarization where at the end an abstract perspective of the model can be presented. Full description of this approach is presented in Chapter 8.

Last but not least, Table 1.2 shows the main features and contributions of each approach to demonstrate the most important features of them. Specific attributes of each approach will be presented in the corresponding chapters. In short, description of each row is as follow:

- **Optimality**: This feature shows whether the obtained solution or solutions are globally optimal.
- **Scalability**: This feature demonstrates ability of the proposed approach in dealing with medium to large problems. For example, Local Search approach is super light and can handle very large instances.
- **Exact**: This attribute shows whether exact alignments (a concept which will be presented later on in this thesis) can be obtained or no.
- **Memory Footprint**: One of the very important metrics for evaluating alignment computation algorithms is considering its memory consumption. This feature quantifies the mentioned metric.
- **Execution Speed**: Apart from memory footprint, this feature shows how fast will be the corresponding execution. Some approaches like genetic algorithms though are memory bounded but having long execution time.
- **Complexity**: This row shows the theoretical complexity of each approach. This feature, though provides an intuition about theoretical complexity of the proposed approaches, it can not be used alone to judge an algorithm, for example, both ILP and Recursive ILP have NP-complete complexities but due to different size of problems they face, the later is much more efficient than the former in practice.

1.4 Related Work

The first and seminal paper regarding the fitness dimension of conformance checking[1] was presented by [68] which introduced *token replay* and also enumerated other dimensions of conformance checking. This approach

[1] From now on for the sake of simplicity when "conformance checking" is mentioned, its fitness dimension is meant.

represents a process model by a Workflow Net (WF-net), and replays the observed trace from an initial marking to the final marking and then based on the remaining and missed tokens quantifies the fitness value. This approach, despite of being fast is sensitive to noise. Paper [7] proposed a cost-based replay technique that measures fitness and takes into account the cost of skipping and inserting individual activities. The technique is based on the A^* algorithm, and can be tailored to answer specific questions (e.g. Does the log conforms to the model? Which events are skipped and inserted?) which ends up with the notion of *alignment*.

An alignment between a process model and corresponding observed trace represents how well the later can be confronted with a trace of the former. An optimal alignment shows the best trace of the model which mimics the observed trace. The main problem with this approach is its high complexity due to calculating alignments, but alignment techniques are more robust with respect to noise rather than replay techniques. More specific, for each observed trace σ in the log, the alignment consists on exploring the synchronous product of model's state space and σ. In the exploration, the shortest path is computed using the A^* algorithm, once costs for model and log moves are defined.

The A^* approach is implemented in ProM [92], and can be considered as the state-of-the-art technique for computing alignments. Several optimizations have been proposed to the basic approach: for instance, the use of ILP techniques on each visited state to prune the search space [5].

Alignment techniques from [5] have been extended recently in [15] for the case of *process trees*, presenting techniques for the state space reduction with *stubborn sets*[2]. Also, high-level deviations are proposed in [5] in form of *deviation patterns* that, as the work in this paper, aim at providing less detailed diagnostics. In spite of the optimizations, Unfortunately, these techniques in [5] cannot handle large inputs. To tackle the mentioned computational challenge, i.e., product of the state-space of a model and the observed trace which is well-known state explosion problem, [31] proposed an alternative for alignment computation of acyclic process models, that encodes the alignment problem as a Constraint Optimization Problem (COP) where COP is a Constraint Satisfaction Problem (CSP) with an objective function which must be optimized with respect to given constraints. The encoding schema is constraints that describe the possible execution order of the transitions in a Petri net (expected behavior), and the order of events in the logs (observed behavior). The COP will find the minimum misalignment between the observed and the expected transitions to compute the corresponding alignment.

Due to the mentioned challenge of alignment computation, some other researchers adopted a decomposition approach. Proposed approaches split

[2] There is no fundamental difference between aligning Petri nets or process trees: only the latter allows for a slightly better memory representation.

the given model, and then conformance checking and alignment computing will be done on each decomposed part separately. The *Refined Process Structure Tree* (RPST), proposed by [97], is a graph parsing technique that provides well-structured parts of a graph. The resulting parse tree is unique and modular, i.e., local change in the local workflow graph results in a local change of the parse tree. It can be computed in linear time using the method proposed in [72], which is based on the *triconnected components* of a given biconnected graph. The proposed approach only works with single sink, single source workflow graphs, which hampers its applicability to real world problems with many sink, source nodes. The work in [63] presents a more efficient way to compute RPST which can deal with multiple source, sink workflow graphs. Based on computed RPST, approaches in [56], [55] and [54] adopt a decomposition approach, in which a given process model is parsed and special structures which are called *Single Entry Single Exit (SESE)* are identified, and then it is decomposed to separate smaller parts for conformance checking of the model and observed behavior. More specific, the observed trace is projected to each SESE and the result will be used for conformance checking of the projected observed trace and corresponding SESE, and this can be done for all SESEs in independently. This technique is very efficient, but the result is decisional (a yes/no answer on the fitness of the trace). Recently, [99] proposed a new approach which provides an algorithm that is able to obtain such an optimal alignment from the decomposed alignments if this is possible, which is called proper optimal alignment. Otherwise, it produces a so-called pseudo-alignment which may not be executable in the net.

Abstraction of business process models is presented in [62]. The core idea is to replace the process fragments inside a given process model with the process tasks of higher abstraction levels, to simplify the given process models for non-technical stakeholders. The key property of the presented approach is *order preservation*, by which the abstraction mechanism ensures that neither new task execution order constraints are produced nor existing ones gone after abstraction. Stated differently, the mentioned property secures the overall process logic to be reflected in the abstracted model. To identify process fragments, the paper uses the notion of *process component* i.e., a process fragment which is connected to the rest of the model by only two nodes namely fragment entry and fragment exit. Identifying process components in a given process model amounts to finding triconnected components of a graph. To this end the presented approach lies on *SPQR-tree* decomposition, by which triconnected components can be obtained. Afterwards, the proposed abstraction rules utilize these components. Four abstraction rules are presented which depend on the structure types returned from the decomposition stage. Since the proposed approach relays on identifying triconnected components of a process model therefore it must have some structural characteristics like being free of self-loop structural patterns and must contain no places with multiple incoming

and multiple outgoing arcs. Similarly, the work in [103] presents *causal behavioural profile* notion for consistency verification between a normative model and its workflow implementation, i.e., to what degree the behavior of the later is captured by the former. The mentioned notion represents a behavioural abstraction that includes dependencies in terms of *order, exclusiveness* and *causality* between pairs of activities of a process model. The general idea of consistency measure is as follows: given the correspondence relation between the sets of transitions of two WF-nets, all respective transitions of two models are aligned, and for each pair of aligned transitions it is checked whether those transitions show the same constraints as defined by the causal behavioural profile. To compute causal behavioural profile efficiently, the presented approach concretises RPST fragments by annotating them with behavioural characteristics. Stated differently, an explicit relation between structural and behavioural characteristics is established.

The work in [65], presents automata-based technique which uses A^* to compute an alignment. In this technique, the process model is transformed to its reachability graph which can be viewed as a automaton, and the event log is converted to a minimal *Deterministic Acyclic Finite State Automaton* (DAFSA), then the two automata are combined into an error-correcting product automaton whose transitions are either a synchronous or asynchronous moves. The created automaton uses three operations corresponding to synchronous move and moves in log or model to synchronize the behavior between DAFSA and reachability graph. Stated differently, nodes in the created automaton represent partial alignments and arcs are operations by which different nodes can be reached. Despite the fact that this method provides all optimal alignments, like state of the art technique in [5], it suffers from state space explosion issue when deals with large models containing nested loops.

Recently other researchers have been tried to apply conformance checking in an online scenario and the rationale is to deal with incomplete observed traces and aim to diagnose deviations when the process is running. In online scenarios, it is not always relevant to mimic the concept of alignments. For example, in very critical environments, alarms must be raised immediately when deviations take place. The first approach to compute conformance checking for on-line data streams in presented in [18]. The core advantage of this technique, with respect to previous off-line approaches, is the ability to check deviations from the de facto behavior in real-time, i.e., immediately after they occurred. This way, possible corrections can be immediately enacted. The work in [93] aims at computing prefix-alignments rather than conventional alignments mentioned earlier. It entails an incremental algorithm that allows for computing both optimal and approximate prefix-alignments in a greedy way which is the result of solving such shortest problem. The approach in [94] takes advantages of decomposition approaches presented earlier and decomposes the given process model into a smaller subprocess models which gain a significant speed-up

when verifying events. Also, by applying decomposition techniques, localizing deviations and volatile parts of the process models becomes more straightforward, allowing end-users to quickly get an insight in which parts of the current model are failing or being violated. Online conformance checking with BPMN formalism for process models is presented in [101] and considers it as a service which in short is called CCaaS. It utilizes basic token replay techniques for online conformance checking due to its conceptual integration with BPMN but as the author mentioned this basic mechanism has a challenge with decision gates, like XOR, hence they proposed token pull mechanism to fix this issue and provides the mentioned service.

Other approaches which made contributions to apply conformance checking without focusing on alignment computations are as follows, the work in [30] given a process model and event logs returns a set of statements in natural language describing the behavior allowed by the model but not observed in the log and vice versa. The method relies on a unified representation of process models and event logs based on a well-known model of concurrency, namely event structures. Specifically, the problem of conformance checking is approached by converting the event log into an event structure, converting the process model into another event structure, and aligning the two event structures via an error-correcting synchronized product. Each difference detected in the synchronized product is then verbalized as a natural language statement. The work in [43] introduced a configurable divide-and-conquer Projected Conformance Checking framework (pcc framework) to compare logs to models and models to models. Instead of comparing the complete behaviour over all activities, the problem is decomposed into comparing behaviour for subsets of activities. For each such subset, a recall, fitness, or precision measure is computed. The averages over these subsets provide the final measures, while the subsets with low values give information about the location in the model/log/system-model where deviations occur. The approach in [14] introduced trace alignment notion for diagnostics in process discovery and conformance checking, in other words the method aligns all the observed traces in the event log to each other with a dynamic programming method inspired from bioinformatics, i.e., Multiple Sequence Alignment, to identify events of an observed trace that are not in accord with general behavior of the same events in other observed traces and uncovers interesting patterns and assists in getting better insights on process executions.

Chapter 2

Preliminaries

This chapter provides the necessary requirements and preliminaries for the presented approaches in the next parts. More specific details will be presented in each part separately. It starts with one of the most well known process modeling formalism, i.e., Petri nets, and explains the related topics to process mining and techniques in this thesis. Following that classical and modern optimization techniques which will be incorporated in proposed approaches are presented. First, the mechanics of Integer Linear Programming (ILP) will be presented, and finally Genetic Algorithm and Heuristic search concepts are explained.

2.1 Process Modeling

Companies and organizations usually use different notations to represent their business processes, which most of them are models made by hand and are not based on a rigorous analysis of existing process data. For example, at the highest abstraction level a process might be represented in Business Process Model and Notation (BPMN), for the sake of understandability for stakeholders and managers who do not want go to the details of the process under consideration. Hand-made processes are delicate and their creation can take a long time. An inadequate model can lead to wrong conclusions and this is why discovering models from event logs can somehow alleviate this issue [3].

With that said, in general a process can be represented in different formalisms, where each of them has various abilities and characteristics. In other words, each process modeling approach is not a completely suitable for everywhere. Thus, appropriate process modeling language selected is especially important. But it must be stressed that often one formalism can easily be translated to other notations and formalisms [80]. It is estimated that there are around 350 process modeling tools, all claiming to support effective, comprehensible, compact, suitable etc. conceptual business modeling [36]. The most well-known and studied formalisms are *Petri nets, C-nets, BPMN, YAWL, EPCs* and *Transittion systems*. Petri nets will be explained in-depth since all the proposed approaches in this thesis are based on this formalism and as mentioned earlier, this is without loss of generality since the results can be translated to other formalisms.

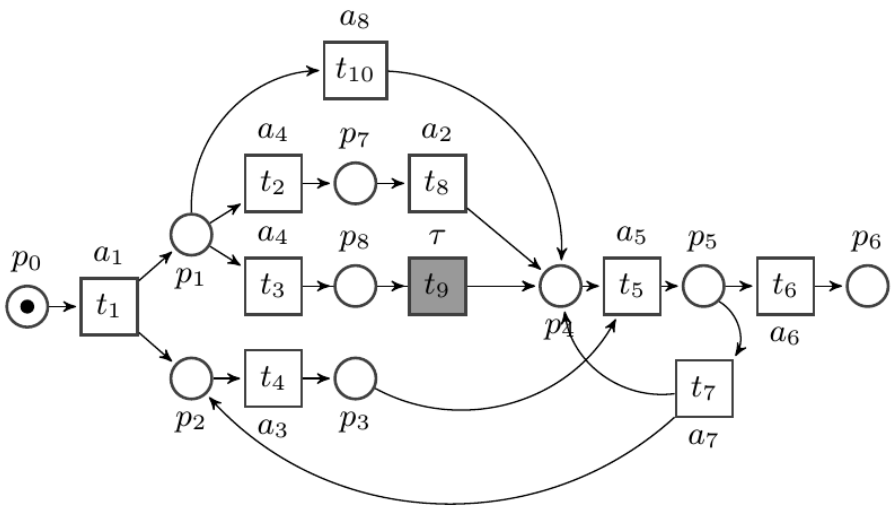


Figure 2.1: Labeled Petri net, transitions are squares, places are circles and tokens are black dots

2.1.1 Petri Nets

Petri nets are similar to block diagrams, flow-charts and networks. They are graphical and mathematical modeling tool applicable to many systems. They can describe systems that are being characterized as concurrent, parallel, asynchronous, distributed and stochastic [57]. Informally, a Petri net is made of two building blocks called *transition* and *place*, and a set of arcs that connects transitions to places and vise versa. The network structure is static, but, governed by the firing rule, *tokens* can flow through the network. For example Fig. 2.1 shows a Petri net where places represent different state of the system and p_0 contains a token. More formally:

Definition 1 (Petri Net)**.** *A* Petri Net *[57] is a 3-tuple* $N = \langle P, T, \mathcal{F} \rangle$*, where* P *is the set of places,* T *is the set of transitions,* $P \cap T = \emptyset$*,* $\mathcal{F} : (P \times T) \cup (T \times P) \to \{0, 1\}$ *is the set of directed arcs which is called flow relation.*

For example in Fig. 2.1, the set of transitions T and places P are $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ and $\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ respectively.

Definition 2 (Labeled Petri Net)**.** *A* labeled Petri net (LPN) *is a 3-tuple* $\langle N, \Sigma, \ell \rangle$*, where* N *is a Petri net,* Σ *is an alphabet (a set of labels) and* $\ell : T \to \Sigma \cup \{\tau\}$ *is a* labeling function *that assigns to each transition* $t \in T$ *either a symbol from* Σ *or the empty symbol* τ*. The set of labeled transitions is represented by* T_ℓ*.*

In Fig. 2.1 the set of labels related to transitions is $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, \tau\}$. Note that t_9's label is τ which shows it is an invisible or silent transition. In this thesis for almost all presented approaches process models are represented by labeled Petri nets. For the sake of simplicity and to avoid complicated figures, labels are only shown when needed.

In response to above definitions, a marked Petri net is a pair (N, m), where $N = (P, T, F)$ is a Petri net and where $m \in B(P)$ is a multi-set over P denoting the marking of the net. Also a *marking* is an assignment of non-negative integers to places. If k is assigned to place p by marking m (denoted $m[p] = k$), we say that p is marked with k tokens. For example in Fig. 2.1 $m[p_0] = 1$ and for other places like p_i, $m[p_i] = 0$.

Definition 3 (Pre-set, Post-set, Sibling)**.** *Given a node* $x \in P \cup T$*, its pre-set and post-set (in graph adjacency terms) are denoted by* ${}^\bullet x$ *and* $x^\bullet$ *respectively and an element with the same pre-set of* x *is called its sibling.*

For example in Fig. 2.1, the pre-set and post set of p_1 are ${}^\bullet p_1 = \{t_1\}, p_1^\bullet = \{t_2, t_3\}$ and for t_7, ${}^\bullet p_5$, $p_4, p_2^\bullet$ respectively. Also t_2 and t_3 as well as t_6 and t_7 are siblings.

The dynamic behavior of a marked Petri net is defined by the so-called *firing rule*. A transition is enabled if each of its input places contains a token. An enabled transition can fire thereby consuming one token from

each input place and producing one token for each output place. More rigorously: a transition t is *enabled* in a marking m when all places in ${}^\bullet t$ are marked. More formally it is denoted by $(N, m)[t\rangle$, iff ${}^\bullet t \leq m$. When a transition t is enabled, it can *fire* or *execute* by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. For example in configuration of Fig. 2.1 only t_1 is enabled since its pre-set i.e., p_0 is marked. When it is fired, it consumes the token in p_0 and puts tokens in $t_1^\bullet$, i.e., p_1 and p_2.

Definition 4 (Reachability Marking)**.** *Given a Petri net N, a marking m' is* reachable *from m, if there is a sequence of firings $t_1 t_2 \dots t_n$ that transforms m into m', denoted by $m[t_1 t_2 \dots t_n\rangle m'$ or $m' \in R(N, m)$. A sequence of transitions $t_1 t_2 \dots t_n$ is a* feasible sequence *if it is firable from the initial marking m_0.*

To derive this concept home consider the model in Fig. 2.1, marking $m'[p_6] = 1$ is reachable from $m[p_0] = 1$ because there is a sequence of activities like $t_1 t_{10} t_4 t_5 t_6$ by which it is possible to reach from the former to the later marking. The set of all reachable marking m from an initial marking m_0 of a Petri net is called *reachability graph* of that model and initial marking. It is not difficult to see that, a marked Petri net may have infinitely many reachable states. Obtaining reachability graphs is easy by simulation and using fast computers but in the case of having infinite reachable state one can resort to *coverability graph*. Creating coverability or reachability graphs can reveal some dynamic behavior properties of the model as follows [57]:

- *k-bounded*: A marked Petri net (N, m_0) is k-bounded if no place ever holds more that k tokens. Formally $\forall p \in P$ and $\forall m \in [N, m_0\rangle$: $m[p] < k$.
- *Liveness*: A marked Petri net (N, m_0) is said to be live if, no matter which marking has been reached from m_0, then for each transition of the model like t there must be a sequence of firing such that the resulted marking enables t. More formally, $\forall m \in [N, m_0\rangle, \exists m' \in [N, m\rangle$ such that $(N, m)[t\rangle$.
- *deadlock free*: A marked Petri net (N, m_0) is said to be deadlock free if at each marking m derived from the initial marking m_0 at least one transition is enabled.
- *Reversibility*: A marked Petri net (N, m_0) is said to be reversible if from each marking $m \in [N, m_0\rangle$, m_0 is reachable from m. This property says that the system can get initial settings after it is executed.

In addition to the above properties, a Petri net is *safe* if it is 1-bounded. To derive these concepts home, the model in Fig. 2.2 (b) is 9-bounded since none of the places can have more than 9 tokens. Also, it is not deadlock free

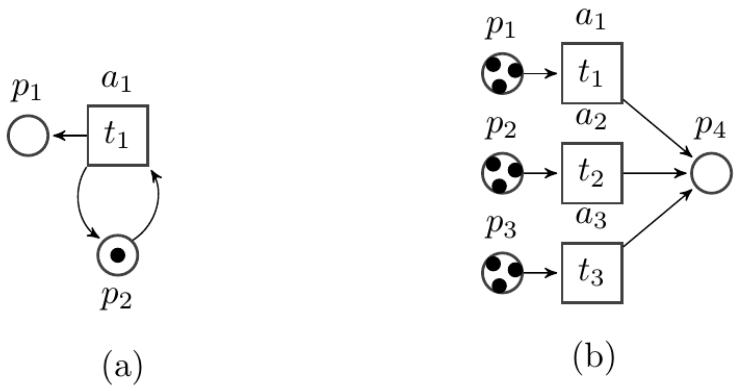


Figure 2.2: Petri nets

since for marking m such that $m[p_4] = 9$ there is no enabled transitions. The model in Fig. 2.2 (a) is unbounded since infinite number of tokens can be placed on p_1 and it is deadlock free since for any marking t_1 is active and can be fired. The model in Fig. 2.1 is safe since the maximum number of tokens a place can have is one also it is not reversible, i.e., initial marking can not be reached when the model is being executed.

Definition 5 (Structural Deadlock)**.** *or simply deadlock in a Petri net is a set of places such that every transition which outputs to one of the places in the deadlock also inputs from one of these places. Formally, a nonempty subset of places P_d of a net N is a deadlock if ${}^\bullet P_d \subseteq P_d^\bullet$, See Fig. 2.3. Deadlocks have the following properties [60], [19]:*

- *If marking $m \in RS(N, m_0)$ is a deadlock state then $P_d = \{p | m[p] = 0\}$, is an unmarked set of places.*
- *Once all of the places in the deadlock become unmarked, the entire set of places will always be unmarked; no transition can place a token in the deadlock because there is no token in the deadlock to enable a transition which outputs to a place in the deadlock.*

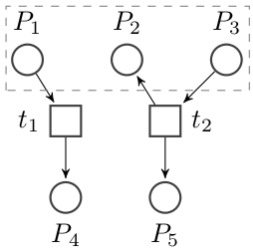


Figure 2.3: $P_d = \{P_1, P_2, P_3\}$, ${}^\bullet P_d = \{t_2\}, P_d^\bullet = \{t_1, t_2\}$

2.1.1.1 Subclasses of Petri Nets

Workflow nets (WF-nets) are subclass of Petri nets which are more suitable to model business processes [84] due to having special places called *Start* and *end*. They are suitable since there could be a mapping from observed traces in the even logs and firing sequences of a WF-net. The definition of WF-nets formally is as follow:

Definition 6 (Workflow Net)**.** A Workflow net (WF-net) *is a labeled Petri net* $\langle N, \Sigma, \ell \rangle$ *where there is a place* start *(denoting the initial state of the system) with no incoming arcs and a place* end *(denoting the final state of the system) with no outgoing arcs, and every other node is within a path between start and end.*

The transitions in a WF-net represent tasks. For the sake of simplicity, the techniques of this thesis assume models are specified with WF-nets. Indeed the model in Fig. 2.1 is a WF-net where p_0 and p_6 are the start and end places. Though WF-nets are appropriate for process representation in process mining tasks as mentioned, they must meet some properties to avoids unexpected results. For example, no deadlocks must be presented, more specific it must be *sound* according to [81]:

Definition 7 (Sound Workflow Net)**.** *A WF-net with input place* start *and output place* end *is* sound *if the following conditions are met:*

- Option to complete: $\forall m \in R(N, start), m[end] \in R(N, m)$ (this property states that from any marking obtained from the initial marking, final marking can be reached)*;*
- Proper completion: $\forall m \in R(N, start)$, *if* $end \in m$ *then only* $m[end] = 1$ (this property says, if the final marking is marked then other places must be without tokens)*;*
- No dead transitions: $\forall t \in T, \exists m \in R(N, start)$ *such that* $(N, m)[t\rangle$.

Fig. 2.1 represents a WF-net which fulfills soundness requirements as just mentioned.

Another important subclass of Petri nets are *Free-Choice Nets (FC-nets)*. FC-nets have the behavioral feature that if two transitions share an input place and it gets marked then both transitions become active, or there is no marking in which one is active and the other is disabled, more formally FC-net is defined as follow [57]:

Definition 8 (Free-Choice Nets)**.** A Free-Choice net (FC-net) *is a Petri net such that every arc from a place is either a unique outgoing arc or a unique incoming arc from a transition, i.e.,* $\forall p_1, p_2 \in P, p_1^\bullet \cap p_1^\bullet \neq \phi \rightarrow |p_1^\bullet| = |p_2^\bullet| = 1$.

According to the above definition an *Extended Free-Choice Net (EFC)*, is a Petri net such that if $p_1^\bullet \cap p_1^\bullet \neq \phi \rightarrow p_1^\bullet = p_1^\bullet, \forall p_1, p_2 \in P$. Also Every

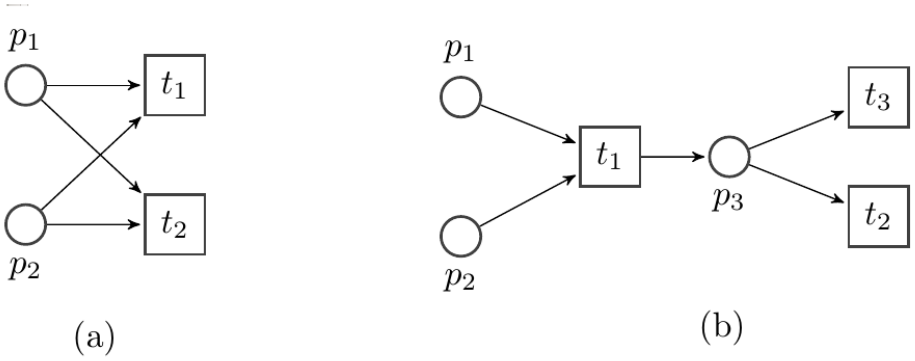


Figure 2.4: (a) Extended Free-choice net, (b) Free-choice net

EFC system can be simulated by an FC system [11]. The models in Fig. 2.4 (a) and (b) are non free-choice and free-choice nets respectively. The model in Fig. 2.4 (a) is not a FC-net since $|p_1^\bullet| > 1$ and $|p_2^\bullet| > 1$. In Fig. 2.4(b), transitions t_2 and t_3 share p_3 as an input place, thus whenever it gets marked both t_2 and t_3 are enabled.

There are some definitions related to FC-nets as follows:

Definition 9 (Clusters)**.** *Let x be a node of a process model (FC-net). The cluster x, denoted by $[x]$ is the minimal set of nodes such that:*

- $x \in [x]$
- *If a place p belongs to $[x]$ then $p^\bullet \in [x]$*
- *If a transition t belongs to $[x]$ then ${}^\bullet t \in [x]$*

The above definition, poses some properties like, the set $\{[x]|x \in (P \cup T)\}$ is a partition of nodes of the corresponding model or in FC-nets if an arbitrary marking enables a transition like t then it enables every transitions of $[t]$ [23].

Definition 10 (Allocations)**.** *Let C be a set of clusters of a model $N = \langle P, T, \mathcal{F} \rangle$, such that every clusters C contains at least one transition. An allocation is a function $\alpha : C \Rightarrow T$ such that $\forall c \in C$, $\alpha(c) \in c$. A transition t is said to be allocated by α if $t = \alpha(c)$ for some cluster c. Also the set of transitions allocated by α is denoted by $\alpha(C)$.*

An allocation α is called *cyclic* if for every cluster $c \in C$, the set $\alpha(c)^\bullet$ contains only places of C [23].
Fig.2.5 shows the set of clusters C by rectangles and some corresponding allocated transitions in green color.

2.1.1.2 Petri Nets and Linear Algebra

Let $N = \langle P, T, \mathcal{F} \rangle$ be a Petri net with initial marking m_0. Given a feasible sequence $m_0[\sigma\rangle m$, the number of tokens for a place p in m is equal to the

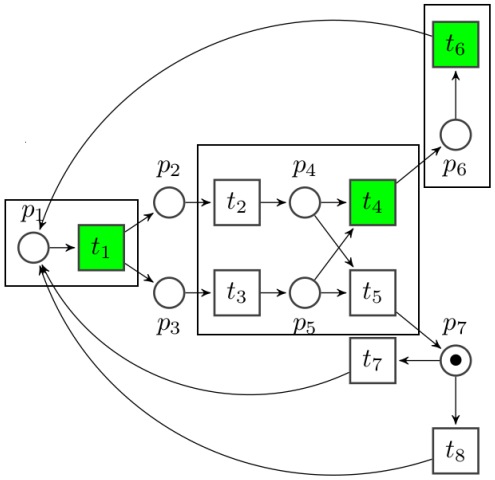


Figure 2.5: Clusters and allocated transitions

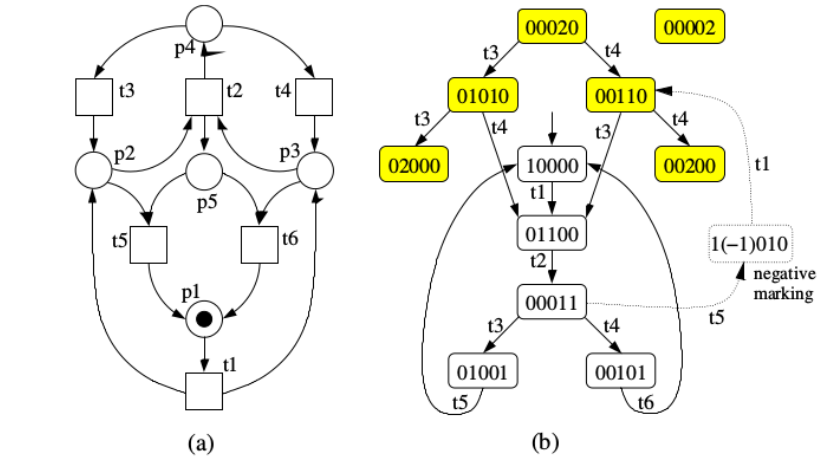
tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the output transitions of p in σ:

$$m[p] = m_0[p] + \sum_{t \in {}^\bullet p} |\sigma|_t \; \mathcal{F}(t,p) - \sum_{t \in p^\bullet} |\sigma|_t \; \mathcal{F}(p,t)$$

The marking equations for all the places in the net can be written in the following matrix form (see Fig. 2.6(c)): $m = m_0 + \mathbf{N} \cdot \widehat{\sigma}$, where $\mathbf{N} \in \mathbb{Z}^{P \times T}$ is the *incidence matrix* of the net: $\mathbf{N}[p,t] = \mathcal{F}(t,p) - \mathcal{F}(p,t)$. If a marking m is reachable from m_0, then there exists a sequence σ such that $m_0[\sigma\rangle m$, and the following system of equations has at least the solution $X = \widehat{\sigma}$

$$m = m_0 + \mathbf{N} \cdot X \tag{2.1}$$

If (2.1) is infeasible, then m is not reachable from m_0. The inverse does not hold in general: there are markings satisfying (2.1) which are not reachable. Those markings (and the corresponding Parikh vectors) are said to be *spurious* [71]. Fig. 2.6(a)-(c) presents an example of a net with spurious markings: the Parikh vector $\widehat{\sigma} = (2,1,0,0,1,0)$ and the marking $m = (0,0,1,1,0)$ are a solution to the marking equation, as is shown in Fig. 2.6(c). However, m is not reachable by any feasible sequence. Fig. 2.6(b) depicts the graph containing the reachable markings and the spurious markings (shadowed). The numbers inside the states represent the tokens at each place $(p_1, \ldots, p_5)$. This graph is called the *potential reachability graph*. The initial marking is represented by the state $(1,0,0,0,0)$. The marking $(0,0,1,1,0)$ is only reachable from the initial state by visiting a negative marking through the sequence $t_1 t_2 t_5 t_1$, as shown in Fig. 2.6(b). Therefore, equation (2.1) provides only a sufficient condition for reachability of a marking and replayability for a solution of (2.1).



$$\underset{m}{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}} = \underset{m_0}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}} + \underset{N}{\begin{bmatrix} -1 & 0 & 0 & 0 & +1 & +1 \\ +1 & -1 & +1 & 0 & -1 & 0 \\ +1 & -1 & 0 & +1 & 0 & -1 \\ 0 & +1 & -1 & -1 & 0 & 0 \\ 0 & +1 & 0 & 0 & -1 & -1 \end{bmatrix}} \underset{\widehat{\sigma}}{\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}$$

(c)

Figure 2.6: (a) Petri net, (b) Potential reachability graph, (c) Marking equation.

For well-structured Petri nets classes equation (2.1) characterizes reachability. The largest class is free-choice, *live*, bounded and *reversible* nets [57]. For this class, equation (2.1) together with a collection of sets of places (called *traps*) of the system completely characterizes reachability [22]. For the rest of cases, the problem of the spurious solutions can be palliated by the use of traps [26], or by the addition of some special places named *cutting implicit places* [71] to the original Petri net that remove spurious solutions from the original marking equation.

Furthermore, even in the case of reachable markings, the marking equation can fail to provide the right Parikh sequence. For instance, the marking $(0, 1, 1, 0, 0)$ is reachable from the marking $(0, 0, 0, 1, 1)$ by firing the sequence $t_4 t_6 t_1$. However, the Parikh vector corresponding to the sequence $t_5 t_1 t_3 t_2$ is also a solution to the marking equation with initial marking set to $(0, 1, 1, 0, 0)$. This observation is important to figure out some issues in the next chapters.

2.2 Process Mining

Process mining aims to discover, monitor and improve real processes by extracting knowledge from event logs readily available in todays information systems [3]. Spectacular growth of event data recently on the one side and maturing process mining techniques on the other side push companies and organizations to exploit process mining from various perspectives.

Starting point for process mining would be an event log and a process model where the later can be modeled in different languages and formalisms. In this thesis as mentioned in the previous section, process models will be modeled using Petri nets and this is without lose of generality. Each event in a log refers to an activity (i.e., a well-defined step in some process) and is related to a particular case (i.e., a process instance). The events belonging to a case are ordered and can be seen as one run of the process. Event logs may store additional information about events. Event logs can be used to conduct different process mining tasks like process discovery, conformance checking and process enhancement.

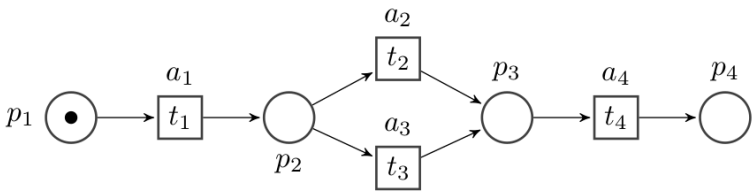


Figure 2.7: Process model

2.2.1 Event Log

To introduce what is an event log, first consider the simple process model in Fig. 2.7. One possibility of an event log is shown in table 2.1

#	Traces
112	$a_1a_2a_4$
159	$a_1a_3a_4$
15	a_1a_4

Table 2.1: Event log

This table shows the event log contains $112 + 159 + 15 = 286$ cases, i.e., instances of some reimbursement process. There are 112 process instances following trace $a_1a_2a_4$ and it contains three activities. Activities are represented by a single character depending on the nature of the given process. For example if the process model in Fig. 2.7 represents an oversimplified

purchasing a product mechanism, then transition of the model could be a_1 ="add to card", a_2, a_3 ="selecting types of card (Master or Visa)" and c ="commit".

Note that events can have all kinds of additional attributes (timestamps, transactional information, resource usage, etc.). Consider for example one of the $843 = 112 * 3 + 159 * 3 + 15 * 2$ events. Let say it is a_1, Such an event refers to the execution of add to card. The event may have a timestamp, e.g., 23-01-2012:8.38, and an attribute describing the resources involved. Moreover, other data attributes (e.g., name of customer or number of loyalty card, type of coupon if there is) and data attributes of the registration event (e.g., a booking reference) may have been recorded. All such attributes can be used by process mining techniques. However, the backbone of process mining is the control-flow perspective. Therefore, for simplicity, events in Fig. 2.7 are described by their activity names only. However, it is good to note that events can have various attributes, e.g., timestamps can be used for bottleneck analysis and resource attributes can be used for organizational mining (e.g., finding allocation rules).

2.2.2 Conformance Cheking

Conformance checking techniques relate events in the log to activities in the model, e.g., events are mapped to transition firings in the Petri net. This way it is possible to compare the observed behavior in the event log and the modeled behavior. For example, one can quantify differences (e.g. to what degree the real execution is in accord with the given process model) and diagnose deviations (e.g., in reality activity a_2 is often skipped although the model does not allow for this). Conformance checking has numerous applications such as:

- For checking the quality of documented processes (asses whether they describe reality accurately),
- To identify deviating cases and understand what they have in common,
- To audit purposes.
- As a starting point for model enhancement.

Conformance checking can be applied for evaluating a process model in four different dimensions, i.e., fitness, precision, generalization and simplicity which are explained in Chapter 1. Conceptually, this evaluation is based on the notion of *alignment* which will be described in the next section. In short, given a process model and observed behavior, an alignment shows how they can be line-up in terms of corresponding event-activity.

2.2.3 Alignment of Observed Behavior and Process Model

Definition 11 (Trace, Event Log, Parikh vector)**.** *Given an alphabet of events* $\Sigma = \{a_1, \ldots, a_n\}$*, a trace is a word* $\sigma \in \Sigma^*$ *that represents a finite sequence of events. An* event log $L \in \mathcal{B}(\Sigma^*)$ *is a multiset of traces*[1]*.* $|\sigma|_a$ *represents the number of occurrences of* a *in* σ*. The Parikh vector of a sequence of events* σ *is a function*$\widehat{\ }\colon \Sigma^* \to \mathbb{N}^n$ *defined as* $\widehat{\sigma} = (|\sigma|_{a_1}, \ldots, |\sigma|_{a_n})$*. For simplicity, we will also represent* $|\sigma|_{a_i}$ *as* $\widehat{\sigma}[a_i]$*. The support of a Parikh vector* $\widehat{\sigma}$*, denoted by* $supp(\widehat{\sigma})$ *is the set* $\{a_i | \widehat{\sigma}[a_i] > 0\}$*. Given a multiset* m*,* $tr(m)$ *provides a trace* σ *such that* $supp(\widehat{\sigma}) = \{x | m[x] > 0\}$*.*

For a trace σ, $\sigma[1], \sigma[2], .., \sigma[k]$ denote its first, second and kth elements respectively. For two Parikh vectors $\hat{\sigma_1}$ and $\hat{\sigma_2}$, $\hat{\sigma_1} \leqq \hat{\sigma_2}$ means that each component of the former is less than or equal to each corresponding component of the later.

Definition 12 (System Net, Full Firing Sequences)**.** *A system net is a tuple* $SN = (N, m_{start}, m_{end})$*, where* N *is a WF-net and the two last elements define the initial and final marking of the net, respectively. The set* $\{\sigma \mid (N, m_{start}) \ [\sigma\rangle (N, m_{end})\}$ *denotes all the full firing sequences of* SN*.*

Definition 13 (Full Model Step-Sequence)**.** *A* step-sequence $\bar{\sigma}$ *is a sequence of multisets of transitions. Formally, given an alphabet* T*:* $\bar{\sigma} = V_1 V_2 \ldots V_n$*, with* $V_i \in \mathcal{B}(T)$*. Given a system net* $N = (\langle P, T, \mathcal{F} \rangle, m_{start}, m_{end})$*, a full step-sequence in* N *is a step-sequence* $V_1 V_2 \ldots V_n$ *such that there exists a full firing sequence* $\sigma_1 \sigma_2 \ldots \sigma_n$ *in* N *such that* $\widehat{\sigma_i} = V_i$ *for* $1 \leq i \leq n$*.*

The main metric considered in this thesis to asses the adequacy of a model in describing a log or observed behavior is *fitness* [85], which is based on the reproducibility of a trace in a model:

Definition 14 (Fitting Trace)**.** *A trace* $\sigma \in \Sigma^*$ *fits* $SN = (N, m_{start}, m_{end})$ *if* σ *coincides with a full firing sequence of* SN*, i.e.,*$(N, m_{start})[\sigma\rangle(N, m_{end})$*.*

Definition 15 (Step-Fitting Trace)**.** *A trace* $\sigma_1 \sigma_2 \ldots \sigma_n \in T^*$ *step-fits* SN *if there exists full model step-sequence* $V_1 V_2 \ldots V_n$ *of* SN *such that* $V_i = \widehat{\sigma_i}$ *for* $1 \leq i \leq n$*.*

As outlined above, the fitness dimension requires an *alignment* of observed trace and model, i.e., transitions or events of the observed trace need to be related to elements of the model and vice versa. Such an alignment reveals how the given trace can be replayed on the process model. The classical notion of aligning event log and process model was introduced by [5]. To achieve an alignment between a process model and an event log, we need to relate *moves* in the observed trace to *moves* in the model. It may be the case that some of the moves in the observed trace can not be mimicked by the model and vice versa.

[1] $\mathcal{B}(\Sigma)$ denotes the set of all multisets of the set Σ.

For instance, consider the model in Fig. 2.7, with the following labels, $\ell(t_1) = a_1, \ell(t_2) = a_2, \ell(t_3) = a_3$ and $\ell(t_4) = a_4$, and the trace $\sigma = a_1a_1a_4a_2$; four possible alignments are:

α_1=
a_1	a_1	$\perp$	a_4	a_2
t_1	$\perp$	t_3	t_4	$\perp$

α_2=
a_1	a_1	$\perp$	a_4	a_2
$\perp$	t_1	t_2	t_4	$\perp$

α_3=
a_1	a_1	a_4	a_2	$\perp$
t_1	$\perp$	$\perp$	t_2	t_4

α_4=
a_1	a_1	a_4	a_2	$\perp$
$\perp$	t_1	$\perp$	t_2	t_4

The moves are represented in tabular form, where moves by trace log are at the top and moves by model are at the bottom of the table. For example the first move in α_2 is $(a_1, \perp)$ and it means that the observed trace (log) moves a_1 while the model does not make any move. Formally an alignment is defined as follow:

Definition 16 (Alignment)**.** *Given a labeled Petri net N and an alphabet of events Σ, Let A_M and A_L be the alphabet of transitions in the model and events in the log, respectively, and $\perp$ denote the empty set, then:*

- (X, Y) *is a* ***synchronous move*** *if $X \in A_L$, $Y \in A_M$ and $X = \ell(Y)$*
- (X, Y) *is a* ***move in log*** *if $X \in A_L$ and $Y = \perp$.*
- (X, Y) *is a* ***move in model*** *if $X = \perp$ and $Y \in A_M$.*
- (X, Y) *is an* ***illegal move****, otherwise.*

The set of all legal moves is denoted as A_{LM} and given an alignment $\alpha \in A^*_{LM}$, the projection of the first element (ignoring $\perp$), $\alpha \downarrow_{A_L}$, results in the observed trace σ, and projecting the second element (ignoring $\perp$), $\alpha \downarrow_{A_M}$, results in the model trace.

For instance consider the previous example, then, $\alpha_1 \downarrow_{A_M} = t_1t_3t_4$ and $\alpha_1 \downarrow_{A_L} = a_1a_1a_4a_2$.

Cost can be associated to alignments, with asynchronous moves having greater cost than synchronous ones [5]. Given assigned cost values, an alignment with optimal cost is preferred.

Definition 17 (Cost of Alignment)**.** *The cost of an alignment can be measured based on its moves. We define distance function $\lambda : A_{LM} \to \mathbb{N}$, as follows where:*

$$\forall (X, Y) \in A_{LM} \qquad \lambda((X, Y)) = \begin{cases} \delta_S & \text{If} \quad X = \ell(Y) \\ \delta_L & \text{If} \quad Y = \perp \\ \delta_M & \text{If} \quad X = \perp \end{cases} \qquad \text{And} \quad 0 < \delta_S < \delta_L, \delta_M \tag{2.2}$$

δ_S is the match cost which represents the cost of a synchronous move, and δ_M and δ_L are penalties for move in model and log respectively or in short asynchronous moves. Therefore the cost of an alignment can be summed over costs of its moves, i.e., $\lambda(\alpha) = \mathbf{\Sigma}_{(X,Y)\in\alpha}\lambda((X, Y))$.

Obviously given an observed trace σ, an alignment, with maximum number of synchronous moves with respect to σ is preferred. If unitary costs are assumed, the goodness of an alignment or *fitness* is the ratio given by the number of events in σ which can be mimicked by the model, i.e., synchronous moves, to the total number of moves in α. The fitness value, is the normalized associate cost, that is a quantity between 0 and 1, thus, the closer fitness value is to 1, the more similar is the model trace to the given observed trace. Formally it is defined as follows.

Definition 18 (Cost Based Fitness Metric)**.** *For a given alignment α, the fitness value, π_α, is defined as follows:*

$$\pi_\alpha = 1 - \frac{(\mathbf{\Sigma}_{(X,\perp)\in\alpha} \times \delta_L + \mathbf{\Sigma}_{(\perp,Y)\in\alpha} \times \delta_M)}{(\mathbf{\Sigma}_{(X,Y)\in\alpha} \times \delta_S + \mathbf{\Sigma}_{(X,\perp)\in\alpha} \times \delta_L + \mathbf{\Sigma}_{(\perp,Y)\in\alpha} \times \delta_M)} \tag{2.3}$$

Def. 18 is the ratio of the total cost of synchronous moves to the whole cost of the alignment, i.e., synchronous and asynchronous moves. It is apparent from Eq. (2.3) that on the occasions when there are no synchronous moves, i.e., $\mathbf{\Sigma}_{(X,Y)\in\alpha} \times \delta_S = 0$, the corresponding fitness value is 0 or $\pi_\alpha = 0$, and it turns out that the model can not mimic the observed trace i.e., maximum deviations. On the other hand if there are no deviations, i.e., $\mathbf{\Sigma}_{(X,\perp)\in\alpha} \times \delta_L = 0$ and $\mathbf{\Sigma}_{(\perp,Y)\in\alpha} \times \delta_M = 0$, then the corresponding fitness value is 1 or $\pi_\alpha = 1$, and it yields the model can mimic 100% the observed trace. As an example given costs $\delta_S = 1, \delta_L = 2$ and $\delta_M = 2$, for α_3 has three asynchronous moves, $\pi_{\alpha_3} = 1 - \frac{2\times2+1\times2}{2\times1+2\times2+1\times2} = 0.25$.

2.2.4 Synchronous Product Petri Net

Given a Petri net and an observed trace, a *synchronous product Petri net*, is a combination of the original model being aligned and a Petri net representation of the (partially ordered) trace in the log. The core alignment question is formalized as follows: Given a synchronous product with a penalty function assigning a non-negative penalty to each transition firing, find a firing sequence from the initial marking to the final marking with the lowest total penalties.

Consider the example model in top of Fig. 2.8 (a). This model is a simple parallelism between transitions B and C after A and before D. Now, consider the trace $\sigma = CD$ translated into a trace net as shown in bottom of Fig. 2.8 (a). Obviously, this trace does not fit the model, as transitions A and B are missing from it. Conceptually, the alignment problem first constructs a so-called synchronous product which is shown in Fig. 2.8 (b). Here, the two black transitions are synchronous combinations of equally labeled transitions in the model and the trace, i.e. they have the same input and output places in both the model and the trace net. The alignment algorithm then finds the shortest execution sequence from the initial state to the final state, where the firing of each transition has an associated cost. Typically, the black transitions, called *synchronous moves*

have the lowest cost, while the model transitions, called *model moves* and the trace net transitions, called *log moves*, have higher costs. For this example, the cheapest firing sequence would be $ABCD$ as depicted in Fig. 2.8 (c). For this alignment, the white transitions A and B have been fired as model moves, and the black transitions C and D have fired as synchronous moves. The marking equation used for the example synchronous product model in Fig. 2.8 (b) is shown accordingly there[2]. Here, the columns corresponding to each transition in the incidence matrix are labeled with m, s, or l for (m)odel, (s)ynchronous, or (log) move.

According to the notion of synchronous product Petri net, optimal alignment can be reformulated. In effect, let $N = \langle P, T, \mathcal{F} \rangle$ be a synchronous product Petri net where $T = T^s \cup T^l \cup T^m$ can be partitioned into sets of transitions corresponding to synchronous moves, log moves and model moves respectively let (N, m_{start}, m_{end}) a corresponding net system with specified initial and final markings. Furthermore let $c : T \to \mathbb{R}^+$ a cost function. An alignment is a full firing sequence $\sigma_a \in \{\sigma \mid (N, m_{start})[\sigma\rangle(N, m_{end})\}$ of this system. Let $c : T \to \mathbb{R}^+$ a cost function, then an optimal alignment is an alignment σ_a such that for all $\sigma \in \{\sigma \mid (N, m_{start})[\sigma\rangle(N, m_{end})\}$ holds that $c(\sigma_a) \leq c(\sigma)$.

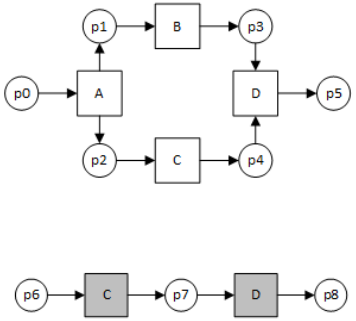
2.3 Optimization Techniques

Optimization is the act of obtaining the best solution under given circumstances or conditions. In design, planning, construction, and maintenance of any engineering system, engineers and scientist have to take many technological and managerial decisions at several stages with respect to corresponding available limitations. The ultimate goal of all such decisions is either to minimize the effort required or to maximize the desired benefit. Since the effort required or the benefit desired in any practical situation can be expressed as a function of certain decision variables, optimization can be defined as the process of finding the conditions that give the maximum or minimum value of a function.

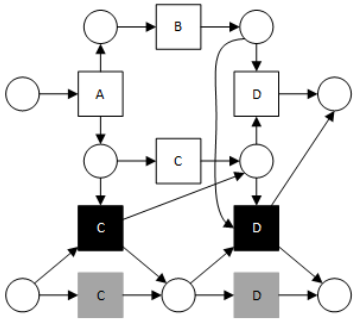
Obviously there is no single method available for solving all optimization problems efficiently. Hence a number of optimization methods have been developed for solving different types of optimization problems. The optimum seeking methods are also known as *mathematical programming* techniques and are generally studied as a part of *operations research*. Operations research is a branch of mathematics concerned with the application of scientific methods and techniques to decision making problems and with establishing the best or optimal solutions. In general a mathematical optimization problem can be stated as follow:

[2] Note that the incidence matrix $\mathbf{N}$ here is decomposed to two matrices. One shows token consumptions and the other token productions.

$$
\begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array}
\begin{array}{c} m_f \\ \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array}\right) \end{array}
=
\begin{array}{c} m_i \\ \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array}\right) \end{array}
-
\begin{array}{c} \\ \left(\begin{array}{cccccccc} A_m & B_m & C_m & D_m & C_s & D_s & C_l & D_l \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \end{array}
.\mathbf{X} +
\begin{array}{c} \\ \left(\begin{array}{cccccccc} A_m & B_m & C_m & D_m & C_s & D_s & C_l & D_l \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right) \end{array}
.\mathbf{X}
$$



(a)



(b)

$\alpha =$

A	C	B	D
$\perp$	C	$\perp$	D

(c)

Figure 2.8: (a) Process model and Petri net of observed trace, (b) Synchronous product Petri net, (c) Optimal alignment

$$\begin{aligned} \text{Find } \mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \text{ which optimizes } f(\mathbf{X}) \\ \text{Subject to the following constraints:} \\ g_i(\mathbf{X}) \leq 0, \qquad i = 1, ..., m \\ l_j(\mathbf{X}) = 0, \qquad j = 1, ..., p \\ \mathbf{X} \in \mathbb{R}^n \end{aligned} \tag{2.4}$$

where $\mathbf{X}$ is an n-dimensional vector called the design vector or vector of variables, $\mathbf{X}$ is called *objective function* which must be minimized or maximized depending on a given problem. $g_i(\mathbf{X}), l_j(\mathbf{X})$ represent the constraints of the given problem and are known as inequality and equality constraints, respectively. The number of variables n and the number of constraints m and p need to be related in any way. A *feasible solution* is a vector like $\mathbf{X}^*$ such that it follows all the constraints, i.e., $g_i(\mathbf{X}^*) \leq 0$ and $l_j(\mathbf{X}^*) = 0, \forall i, j$. It is called *optimal feasible solution* if for all feasible solutions like $\mathbf{X}$, $f(\mathbf{X}^*) < f(\mathbf{X})$ if the objective function must be minimized and vise versa.

The problem stated in Eq. 2.4 is called a *constrained optimization* problem since the objective function will be optimized with respect to some constraints. If there are no constraints then it will be called a *non-constraint optimization* problem. Also optimization problems can be categorized based on types of variables involved and forms of objective functions and constraints. For example, if the objective function as well as constraints, i.e., $f(\mathbf{X})$ and $g_i(\mathbf{X})$, $l_j(\mathbf{X})$ are linear in terms of variables then it is called *linear programming*. Linear programming is by far the most widely used method of constrained optimization. The largest optimization problems in the world are LPs having millions of variables and hundreds of thousands of constraints. With recent advances in both solution algorithms and computer power, these large problems can be solved in practical amounts of time. Also, a problem with integer variables is called *integer programming* and by considering linearity assumption it is *integer linear programming*. All optimization problems in this thesis are of the later forms.

The modern optimization methods, also sometimes called nontraditional optimization methods, have emerged as powerful and popular methods for solving complex engineering optimization problems in recent years due to emerging powerful computing machines and methods. These methods include *genetic algorithms, simulated annealing, particle swarm optimization, ant colony optimization, neural network-based optimization*, and *fuzzy optimization*. The genetic algorithms are computerized search and optimization algorithms based on the mechanics of natural genetics and

natural selection. The genetic algorithms were originally proposed by John Holland [35]. The simulated annealing method is based on the mechanics of the cooling process of molten metals through annealing. The method was originally developed by Kirkpatrick, Gelatt, and Vecchi [40]. local search is a heuristic method for solving computationally hard optimization problems. Also, *Local search* is an iterative method of optimization. It can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

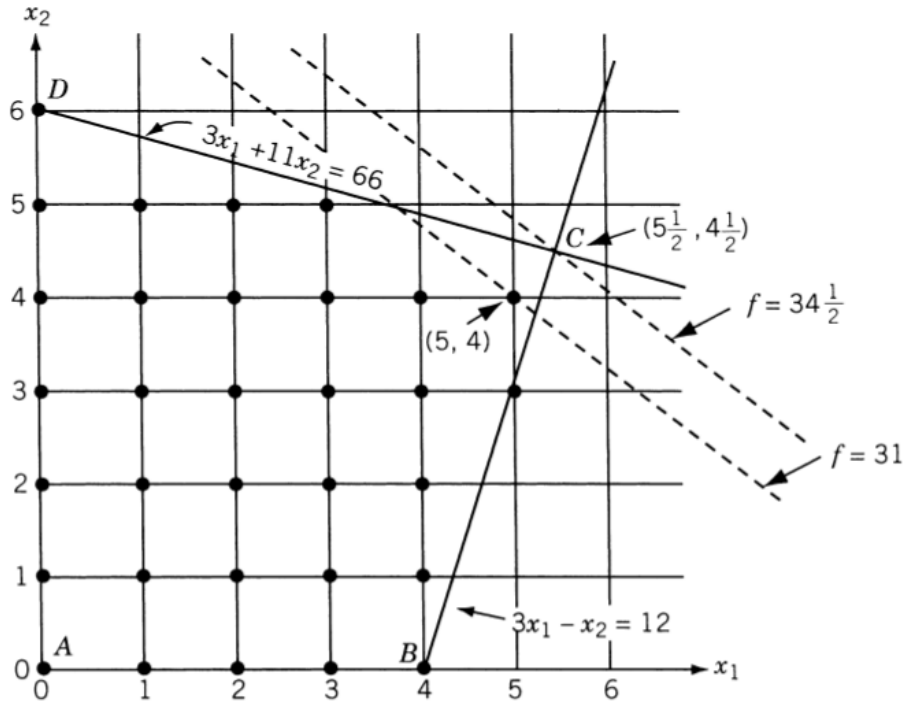


Figure 2.9: Graphical solution of Eq. 2.5 [2]

2.3.1 Integer Linear Programming

In all optimization techniques, variables can be of any types, i.e., continues or integer. When all the variables are constrained to take only integer values in an optimization problem, it is called an *all-integer programming problem* [2]. As mentioned above if the objective function and constraints are linear in terms of involved variable it is called Integer Linear Programming (ILP). An example which has variables x_1 and x_2 is as follow:

$$\begin{aligned} \text{Find } \mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ which maximize } f(\mathbf{X}) = 3x_1 + 4x_2 \\ \text{Subject to the following constraints:} \\ 3x_1 - x_2 \leq 12 \\ 3x_1 + 11x_2 \leq 66 \\ x_1, x_2 \geq 0, x_1 \text{ and } x_2 \text{ are integers} \end{aligned} \tag{2.5}$$

The graphical solution is demonstrated in Fig. 2.9, where the constraints are shown by solid lines and values of the objective function by dashed lines. One sees that point C is not a feasible solution since it violates the constraint of having integer solutions. Indeed it would be the optimal feasible solution if the variables were continues. The optimal feasible solution where the variables are integer as demonstrated is $(x_1 = 5, x_2 = 4)$. It must be mentioned despite of the fact that linear programing problems have polynomial complexity or P, ILPs are *NP-complete* [71], which means that the problem can be solved in polynomial time using a Non-deterministic Turing machine.

2.3.2 Heuristic Search

Solving a problem, usually means looking for one or some solutions, which will be the best among others. The whole space of all candidate solutions (it means objects among those the desired solution is) is called *search space* (or *state space*). Each point in the search space represent one feasible or candidate solution. Each feasible solution can be *scored* by its value or fitness for the given problem. Definitely the desired solution is a point in the given search space. The search for a solution is then equal to looking for some extreme (minimum or maximum) in the corresponding search space. The search space can be identified totally by the time of solving a problem, but in most cases only a few points are known to us and other points,i.e., candidate solutions, will be generated as the process of finding solution continues.

The grand challenge which will be faced is that the search can be very complicated in most real world problems. One does not know where to look for the solution and where to start. Usually one must consider all the points in the search space to find the best solution. This way of search space exploration is called *exhaustive search*, i.e., it checks each and every solution in the search space until the best global solution is found, meaning that there is no way to be sure that you have found the best solution using exhaustive search unless you examine every point in the search space.

There are many methods on how to find some suitable solutions (ie. not necessarily the best solution). *Hill climbing, tabu search, simulated annealing* and *genetic algorithm* are those methods which patrol the search space with different methodologies to find suitable solutions. The solution

found by these methods is often considered a good but not the best solution, because it is not often possible to prove what is the real optimum.

To help for finding desired solutions in the given search space usually some general techniques and domain knowledge can be injected to the algorithm under consideration. Usually *Heuristics* are methods that use intuition or common sense approaches to solve a problem. They help algorithms in searching solutions to not consider all points in the search space and only focus on areas where the chance of having good solutions are higher than other areas. Obviously bad heuristics results in an almost exhaustive algorithms. *Heuristic algorithms* usually are not expected to find the best answer to a problem, but are only expected to find solutions that are "close enough" to the best.

2.3.3 Local search

Local search is a kind of heuristics for solving computationally hard optimization problems. In other words, instead of exhaustively searching the entire space of possible solutions or points, the attention can be focused within a local neighborhood of some particular solution. This procedure is explained in the following steps:

1. Generate a solution from the search space. It can be a random guess or can be obtained by some other methods. Evaluate the merit of the chosen solution and define this as the current solution.

2. Apply modifications to the current solution to generate a new solution and evaluate its merit.

3. If the new solution is better than the current solution then exchange it with the current solution; otherwise discard the new solution and do another modification different from the previous one.

4. Repeat steps 2 and 3 until no transformation in the given set improves the current solution.

The key to understanding how the local search algorithm works lies in the kind of modifications applied to the current solution. There are two extremes:

- At one extreme, the modifications could be defined to return a potential solution from the search space selected uniformly at random. In this case, the current solution has no effect on the probabilities of selecting any new solution, and in fact the search becomes essentially enumerative. In the worst case, it's possible that one might resample points that were already tried which is completely inefficient.

- At the other extreme, the modifications always return the current solution and this gets you nowhere. In other words no improvement can be achieved.

Indeed one can take advantage of local searches where modifications lie between two extremes. In that way, the current and starting solution definitely imposes a bias on where the next search can be done, and when something better can be found and current solution can be updated. If the size of the neighborhood is small then neighborhood can be searched quickly but it comes at the cost of getting trap at a local optimum. In contrast, if the size of the neighborhood is very large then there is less chance of getting stuck, but the efficiency of the search may suffer.

Algorithm 1 Local Search

```
1: t ← 0
2: Initialize best
3: repeat
4:     local = 0
5:     Select a current point V_c at random
6:     Evaluate V_c
7:     repeat
8:         N(V_c) =Select all new points in the neighborhood of V_c
9:         Select point V_n ∈ N(V_c), with the best evaluation score, using
10: eval(V_n)
11:        if eval(V_n) is better than eval(V_c) then
12:            V_c ← V_n
13:        else
14:            local ← 1
15:        end if
16:    until Local
17:    t = t + 1
18:    if V_c is better than best then
19:        best ← V_c
20:    end if
21: until t = MAX
```

2.3.4 Hill-climbing

These methods, are all local search methods, which use an iterative improvement technique. The technique is applied to a single point, i.e., current solution, in the search space. During each iteration, a new point is selected from the neighborhood of the current point. If that new point provides a better value in light of the evaluation function, the new point becomes the current point. Otherwise, some other neighbor is selected and compared against the current point. The method terminates if no further improvement is possible, or it runs out of time or meets some stopping criteria. It is clear that hill-climbing approaches find global optimum and there is no guarantee on finding globally best solutions. Moreover, it is not easy and most of the times impossible to evaluate error of the obtained solution or bounding the relative error with respect to the best solution, i.e., global optimum, since it is unknown. Given the problem of converging on

only locally optimal solutions, we often have to start hill-climbing methods from a large variety of different starting points. The hope is that at least some of these initial locations will provide a path that leads to the global optimum. Initial solutions or points must be chosen at random, or on some grid or regular pattern, or even in based on other information that's available, perhaps as a result of some prior search or some super light methods which approximate better solutions rather than random guesses.

Alg. 1 represents the general schema of hill-climbing as a local search. For a initial point $\mathbf{V_c}$ all possible neighbors of it are considered. Among them, the one with best fitness value, i.e., $\mathbf{V_n}$, is selected to compete with $\mathbf{V_c}$ in terms of evaluation score (line 11). If the former was better than the later in terms of fitness value then the algorithm jumps to $\mathbf{V_n}$ and consider it as the current solution. Otherwise, no local improvement is possible and the algorithm has reached a local or global optimum (line 14). In such a case, the next iteration (line 17) of the algorithm is executed with a new current solution is selected at random. In other words, the algorithms jumps to another area of search space with the hope of finding better solution than ones found so far.

2.3.5 Best-First Search (A^*)

Greedy algorithms like local searches do not always perform very well. The problem is that the obtained solution at one time might not be the best one at a later time, which is kind of trap. But if there was an evaluation function that was sufficiently informative to avoid these traps, these greedy methods could be used in a better way and taking advantage as much as we can. This simple idea leads to a concept called *best-first* search, and its extension, the A^* algorithm. A^* is a kind of algorithm for exploring a given search space. It is an extension of Dijkstra's algorithm for finding the shortest path and proposed in [34]. If the search space can be organized as a tree, then the challenge is how it must be parsed. The intuitive ways are parsing the tree using depth-first or breadth-first alongside with backtracking strategies to parse the provided tree. The best strategy regardless of those just mentioned is to order the available nodes according to some heuristic that corresponds with our expectations of what we will find when we get to the deepest level of our search. The aim is to search first those nodes that offer the best chance of finding something good. The schema of best-first search strategy is shown in Alg. 2, [51]:

Algorithm 2 Best-First

```
1: Input: v                                   ▷ A node in the graph
2: for each available w do                    ▷ w is an available node from v
3:     Assign a heuristic value to w          ▷ Evaluating the available nodes
4: end for
5: q ← Best available w                       ▷ Selecting the best available node
6: Best-First(q)
```

Alg.2 has two lists of nodes, *open* and *closed*. The former indicates all of

the available nodes, whereas the latter shows those nodes that met already. When the best-first procedure starts from a particular node, which is placed on the closed list, all its children are moved to the open list (i.e., they become available). They are evaluated by some heuristic and the best is selected for further processing. It is worth mentioning that if the search space is a general directed graph, one of the children of a node might already be present in the open or closed lists. If that happens, the node that met already must be re-evaluated again and move it from the closed list to the open list.

The main differences between uninformed searches, i.e., depth-first search and breadth-first, and best-first search are

- Best-first search explores the most promising next node, i.e., the algorithm hopes that from that node there is a path to the global optimum. Depth- first search goes as deep as possible in an arbitrary pattern and breadth- first search explores all the nodes on one level before moving to the next.
- Best-first search uses a heuristic that provides a merit value for each node, whereas depth-first search and breadth-first search do not. Stated differently, best-first search look at the future whereas depth and breadth searches do not. Best-first search uses a heuristic that provides a merit value for each node, whereas depth-first search and breadth-first search do not.

Obviously, The efficiency of a best-first algorithm is heavily based on the adopted heuristic. More precisely a best-first algorithm, evaluates a node like v as follow:

$$eval(v) = c(v) + h(v) \tag{2.6}$$

Where the first term in Eq. 2.6 denotes the cost needs to reach v from an initial point in the corresponding search space and the second term shows the estimated cost to reach the desired solution from v. The main challenge here is how to estimate the true cost for the future path inherits from v. This is the place that heuristic functions play an pivotal role. Indeed the quality of heuristics in best-first search can be judged in terms of *admissibility*. A heuristic is admissible if it underestimate the true cost. More formally let $h^*()$ be the function such that if it used in Eq. 2.6, then we get global optimum. It is clear that there is no oracle to give us $h^*()$, therefore a heuristic function like $h()$ is admissible if for all nodes underestimate the true cost, i.e., $\forall v, h(v) \leq h^*(v)$. The aforementioned condition, guarantees that an admissible heuristic will always find global optimum.

2.3.6 Genetic Algorithm

Genetic algorithms (GAs) are the earliest, most well-known, and most widely- used evolutionary algorithms. GAs are simulations of natural se-

lection that can solve optimization. GAs are usually used for function optimization problems but, they comprise a much more broad class of systems than function optimizers. One can use GAs to study the dynamics of adaptive systems, to provide advice to fashion designers and for many other non-optimization applications [52]. Sometimes making a clear line between an optimization algorithm and a non-optimization algorithm is impossible because all algorithms attempt to function as well as possible. Genetic Algorithms are kind of randomized in nature, but they perform much better than random local or blind search but it comes at the price of injecting the application information accurately.

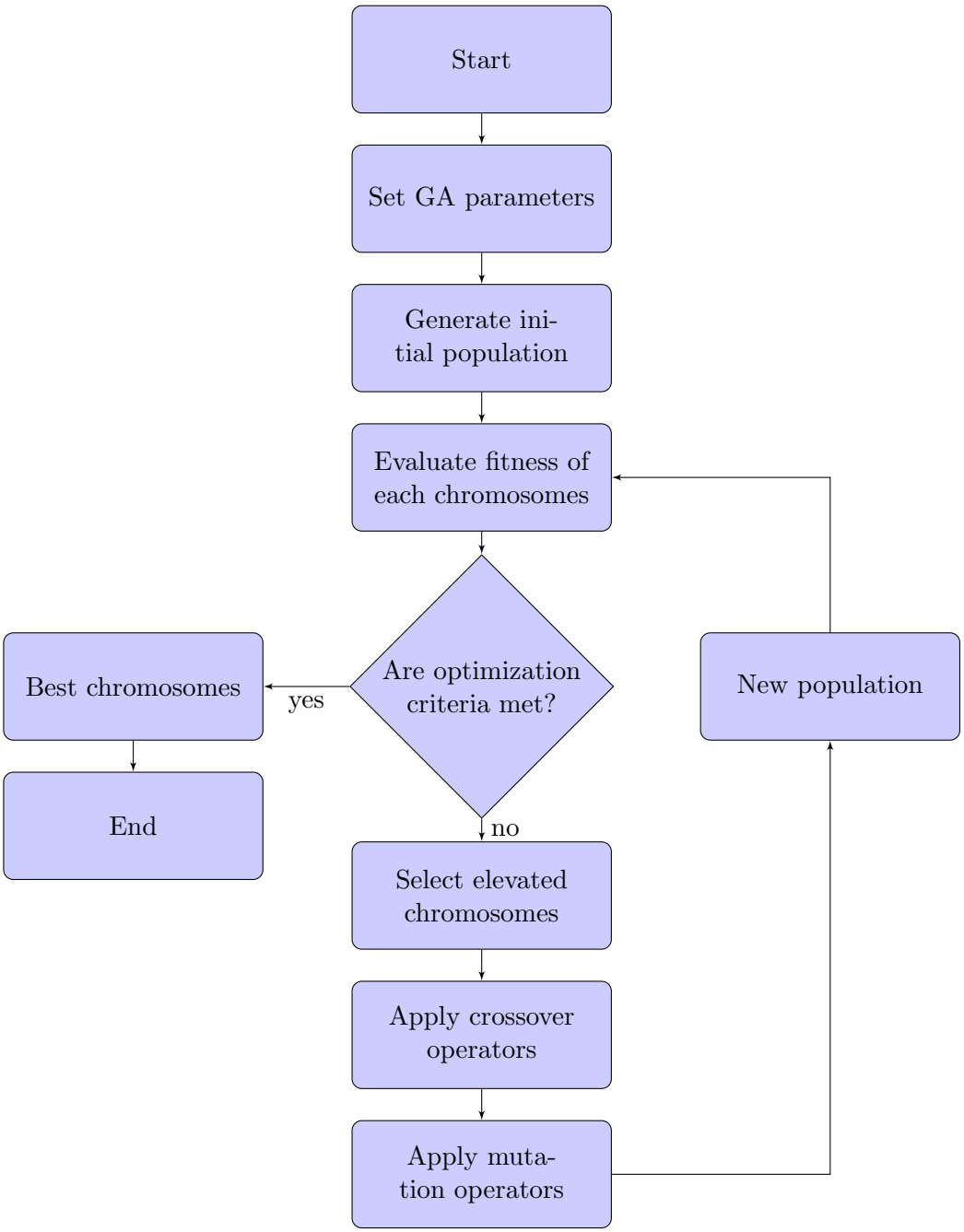
Before moving forward, some basic terminologies which will be used shortly, are explained in formally here:

- **Chromosomes**: A chromosome is one candidate solution to the given problem.
- **Gene**: A gene is one element position of a chromosome.
- **Allele**: The value a gene takes for a particular chromosome.
- **Population**: It is a subset of all the possible (encoded) solutions or chromosomes to the given problem. Obviously the greater the population is, the higher chance of obtaining the desired solution.

GAs use a direct analogy of natural behavior in the real world. They work with a population of *individuals* or chromosomes where each represent a candidate solution to the given problem. Each candidate solution receives a fitness score which denotes how good is that with respect to the problem. Different fitness values can be defined according to the given problem. The highly fit individuals have more chances to mate with others which result in *offspring* that share some features in terms of genes from each parents. Finally, low fit individuals have less chances of surviving in the next generations. A new population of possible solutions is then produced by selecting best individuals from the current generation and mating them to produce new set of individuals. The new population of individuals in average have better fitness value than previous one. In this way over many generations good characteristics spread over the population. By mating individuals with high fitness scores the most promising area of search space will be explored. If the genetic operators designed well, then GA will converge to optimal solution or very close to it. This procedure in flowchart is depicted in Fig. 2.10. The details are as follows:

- **Set GA parameters**: In this step, different parameters which are heavily based on the given problem, like encoding of the given problem, fitness function, and required operators are defined. There are different types of encoding which are explained shortly as follows:

Figure 2.10: GAs flowchart



1. **Binary encoding**: In binary encoding every chromosomes are represented by strings of 0 and 1. Binary encoding is the most used encoding schema. Knapsack problem is the representative one that can be solved easily with tis schema. Indeed each bit says, if the corresponding item is in knapsack.
2. **Permutation encoding**: In this encoding each chromosome is represented by a string of numbers which denotes different permutation of a set. This kind of encoding is useful for ordering problems. Traveling Sales Man (TSP) problem is a typical example for solving with this encoding, where each chromosome shows the order of cities which will be visited [64]. This encoding schema are used in one of the contributions in this thesis, Chapter 6.
3. **Value encoding**: This kind of encoding can be used in problems, where some complicated value, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. That is to say, this encoding is well suited where the objective function is continues. The most representative problem for this encoding is weights optimization in learning Neural networks. Real values in chromosomes represent corresponding weights [53].
4. **Tree encoding**: Tree encoding is used mainly for problems like evolving programs or expressions which is called genetic programming. A representative example is finding a function given inputs and outputs.Stated differently some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs. Each chromosome represent that function by a tree. Also, this kind of encoding can be used in making tree-based classifiers. Indeed they are important in pattern recognition and have been well studied. Although the problem of finding an optimal decision tree has received attention, it is a hard optimization problem [37].

This part is the most important one in every GA, since it can completely affect the results.

- **Generate initial population**: This step, generates chromosomes or individuals according to the parameters defined in the previous step.

- **Evaluation fitness of each chromosome**: This step evaluates the merit of each candidate based on the defined fitness function in the fist step. It must be emphasized that this function which can be composed of some other functions that must be super light in terms

of computation since they are applied to each chromosome in every generation.

- **Are optimization criteria met?**: This step is the one which states whether chromosomes of the final population are enough good. There are many criteria which can be considered to terminate for the future generations, like convergence of the difference between average fitness values of two consecutive populations or lacking the diversity among chromosomes, i.e., all or most of chromosomes in the current population are identical. If termination is decided then high elevated chromosomes are selected as the best solutions.

- **Select elevated chromosomes**: If termination is not decided then chromosomes with good fitness values will be selected, it is clear that chromosomes with higher fitness values have more chances of getting survived for the next generation of population. There are some selection methods regarding this issue with different policies, like *Tournament Selection, Rank Selection, Roulette Wheel Selection* and *Boltzmann Selection*. A good survey which compares different selection methods is presented in [13].

- **Apply crossover operators**: After selecting elevated chromosomes, its time to apply different operators over them. This will allow us to explore various parts of search space and navigate the algorithm to areas where better solutions are resided. Crossover operators are usually apply to a pair of chromosomes. In fact a chromosome with high fitness score has more chances to mate with other chromosomes. The resulted offspring might be better than their parents in terms of fitness value or vice versa. The objective of these operators is to select good parts of each parent to make better offspring. The success of these operators result in goof convergence of solutions but if they are not design well according to the given problem, GAs become super slow and inefficient.

- **Apply mutation operators**: This operator receives one operand, i.e., a chromosome, and does some modifications over operand's genes. The modification is based on the encoding problem, for example in binary encoding it can be changing 0 to 1 or vice versa. The objective of this operator is jumping to a new area of search space randomly. Thus the probability of applying this operator should be kept very short since otherwise it makes the whole algorithms as a random search algorithm since jumping randomly in the search space never allows convergence. Indeed this operator is a way to get rid of local optima in the corresponding search space.

Whenever the above steps are done, there is a new population of chromosomes which in average have better fitness scores with respect to past

generations. The iteration continues until a satisfaction criteria is met or convergence happens.

GAs are usually simple and robust [10]. Some advantages of GA are as follows:

- It can be faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- It can be use in both discrete and continuous domains as well optimizing multi-objective functions very well.
- Providing a list of good solutions and not just a single solution.
- Always gets an answer (not the best one always) to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Despite of having diverse and enormous applications and advantages just mentioned, GAs also have some limitations as follows:

- Lack of guarantee for finding global optimal solutions.
- GAs might be computationally expensive to implement.
- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.

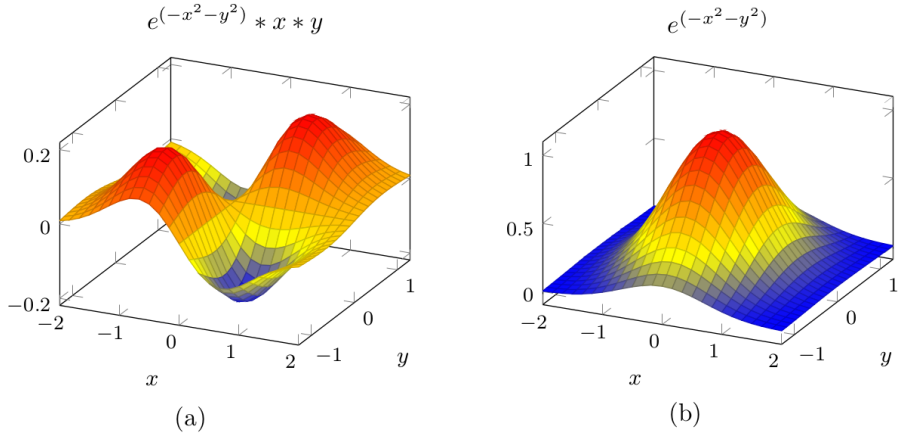


Figure 2.11: Surface of (a) a multi-modal and not convex function, (b) Convex function

Usually real-world function optimization problems often pose nonlinear constraints, objective functions which are not convex and incorporate noisy observations or random processing, or include other conditions that do not

conform well to the prerequisites of classic optimization techniques. thus, in addition to advantages and disadvantages that just described, GAs can be compared with traditional optimization techniques, for example ILP and LP, in terms of surface optimization, see Fig. 2.11(a), (b). More precisely, the response surfaces, i.e., the objective function, posed in many real-world problems are often multi-modal, i.e., having many local optima, and gradient-based methods rapidly converge to local optima which may yield insufficient performance. For simpler problems, where the response surface is strongly convex[3], GAs do not perform as well as traditional optimization methods in terms of efficiency [8], in fact the later take advantages of analytical solutions and it is not a surprising fact because as these techniques were designed to take advantage of the convex property of such surfaces. Some works did empirical comparisons among applying classical methods, i.e., linear programming, and GAs to multi-modal functions, where the former posed significant advantage [70] in terms of the found solutions.

In this thesis, the main interest in GAs is their specific application as optimization algorithms.

[3] A function surface is convex if a local optima exists then it is the global optima.

Part II

Classical Optimization Approaches

This part of the thesis is centered around alignment computation methods which are based on Integer Linear Programming (ILP) as an optimization tool. This part starts with a method which fully takes advantage of structural theory of Petri nets and the marking equation i.e., Eq. 2.1 in Chapter 2, to obtain an approximate alignment. Following that another method will be presented, which combines ILP with A^* method to avoid some deficiencies of the former approach. Indeed, the later approach provides a trade-off between complexity and quality of obtained solutions.

Contribution: The proposed approaches of this part, i.e., Chapters 3, 4 were published in *International Conference on Business Process Management (BPM 2016)* [74] and *International Conference on Advanced Information Systems Engineering (CAISE 2017)* [90] respectively as mentioned in Table 1.1 in Chapter 1.1, Sect. 1.3. Also those approaches have been implemented in Python (ALI [73]) and Java (ProM [92]).

Chapter 3

Monolithic Integer Linear Programming

3.1 Introduction

This chapter presents a technique to compute a particular type of alignments, called *approximate alignments*. In an approximate alignment, the granularity of the moves is user-defined (from singletons like in the original definition of alignments, to non-unitary sets of activities), thus allowing for an abstract view, in terms of *step-sequences*, of the model capability of reproducing observed behavior. The implications of generalizing the concept of alignment to non-singleton steps are manifold: conformance checking techniques can be discretized to a desired (time) granularity, e.g., when the ordering of activities in a period is not important for the diagnosis. Also, other techniques like *model repair* [28] may be guided to only repair coarse-grain deviating model parts. Finally, in domains where a fine-grained ordering of activities is not needed approximate alignments can play an important role (e.g., health care [45]).

It is assumed that the input models to be specified as Petri nets. This is without loss of generality, since there exist transformations from other notations to Petri nets. Given a Petri net and a trace representing the observed behavior, we use the structural theory of Petri nets, Sect. 2.1.1, and materials presented in Chapter 2 to find an approximate alignment. This means that at the end we solve Integer Linear Programming (ILP) models whose resolution provide a model firing sequence that mimics the observed behavior. Importantly, these ILP models are extended with a cost function that guarantees (under certain structural conditions on the process model) a global optimality criteria: the obtained firing sequence is mostly similar to the observed trace in terms of the number of firings of each transition. This optimality capability represents one clear difference with respect to current distributed approaches for conformance checking which focus on the decisional problem of checking fitness, but not to compute optimal alignments [86, 56].

The organization of this chapter is as follow. The formalization of approximate alignments will be presented in Sect. 3.2. Sect. 3.3 describes the overall framework of ILP encoding for computing approximate alignments which consists of two steps. Sects. 3.4, 3.5 detail the mentioned two steps. Finally, Sect. 3.6 provides advantages and disadvantages of the presented approach.

3.2 Approximate Alignment of Observed Behavior

As outlined in Sect. 2.2.3, the fitness dimension requires an *alignment* of trace and model, i.e., transitions or events of the trace need to be related to elements of the model and vice versa. Such an alignment reveals how the given trace can be replayed on the process model. The classical notation of aligning event log and process model was presented in Definition 16. As a reminder an alignment between process model and event log relates moves in the trace to moves in the model. It may be the case that some of the moves in the trace can not be mimicked by the model and vice versa, i.e., it is impossible to have synchronous moves by both of them.

In this chapter a different notion of alignment is introduced. In this notion, denoted as *approximate alignment*, moves are done on multisets of activities (instead of singletons, as it is done for the traditional definition of alignment). Intuitively, this allows for observing step-moves at different granularities, from the finest granularity ($\eta = 1$, i.e., singletons) to the coarse granularity ($\eta = |\sigma|$, i.e., the Parikh vector of the model's trace).

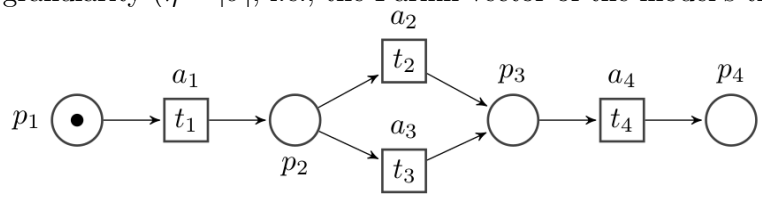


Figure 3.1: Process model

To illustrate the notion of approximate alignment, consider the process model in Fig. 3.1 and trace $\sigma = a_1a_1a_4a_2$. Some possible approximate alignments with different level of granularities are:

$$\alpha_1 = \begin{array}{|c|}\hline \{a_1, a_1, a_4, a_2\} \\ \hline \{t_2, t_1, t_4\} \\ \hline \end{array} \quad \alpha_2 = \begin{array}{|c|c|c|}\hline a_1 & a_1 & \{a_4, a_2\} \\ \hline t_1 & \bot & \{t_4\ , t_2\} \\ \hline \end{array}$$

$$\alpha_3 = \begin{array}{|c|c|c|c|c|}\hline a_1 & a_1 & a_4 & a_2 & \bot \\ \hline \bot & t_1 & \bot & t_2 & t_4 \\ \hline \end{array}$$

For instance, approximate alignment α_2 computes a step-sequence $t_1\{t_4, t_2\}$, meaning that to reproduce σ, the model first fires t_1 and then the step $\{t_4, t_2\}$ is computed, i.e., the order of the firings of the transitions of this step is not specified.

Definition 19 (Approximate Alignment)**.** *Given a labeled Petri net N, let A_M and A_L be the set of transitions in the model and events in the log, respectively, and $\perp$ denote the empty multiset.*

- (X, Y) *is a* ***synchronous move*** *if $X \in \mathcal{B}(A_L)$, $Y \in \mathcal{B}(A_M)$ and $\ell(Y) = X$*
- (X, Y) *is a* ***move in log*** *if $X \in \mathcal{B}(A_L)$ and $Y = \perp$.*
- (X, Y) *is a* ***move in model*** *if $X = \perp$ and $Y \in \mathcal{B}(A_M)$.*
- (X, Y) *is a* ***approximate move*** *if $X \in \mathcal{B}(A_L)$, $Y \in \mathcal{B}(A_M)$, $X \neq \perp$, $Y \neq \perp$, $X \neq \ell(Y)$, and $X \cap \ell(Y) \neq \perp$*
- (X, Y) *is an illegal move, otherwise.*

The set of all *legal* moves is denoted as A_{LM}. Given a trace σ, an approximate alignment is a sequence $\alpha \in A_{LM}^*$. The projection of the first element (ignoring $\perp$ and reordering the transitions in each move as the ordering in σ) results in the observed trace σ, and projecting the second element (ignoring $\perp$) results in a step-sequence.

Moves in an approximate alignment can be assigned with different costs similar to Definition 17. For a given trace different alignments it can be defined with respect to the level of agreement with the trace.

Definition 20 (Cost of Approximate Alignment)**.** *The cost of an approximate alignment can be defined based on its moves, i.e., step-sequences. This distance function is $\Psi : \mathcal{B}(A_L) \times \mathcal{B}(A_M) \to \mathbb{N}$ where:*

$$\forall (X, Y), \Psi(X, Y) = |X \Delta \ell(Y)| \tag{3.1}$$

although other possibilities could be considered as a distance function.[1]

For example $\Psi(\alpha_2) = \Psi(\{a_1\}, \{\ell(t_1)\}) + \Psi(\{a_1\}, \perp) + \Psi(\{a_2, a_4\}, \{\ell(t_2), \ell(t_4)\})$ $= 0 + 1 + 0 = 1$. For the other approximate alignments $\Psi(\alpha_1) = 0$ and $\Psi(\alpha_2) = 3$. Notice that the optimality (according to the distance function) of an approximate alignment depends on the granularity allowed.

3.3 Structural Computation of Approximate Alignments

Given an observed trace σ, in the presented approach computes approximate alignments using the structural theory introduced in Subsection 2.1.1.2. The technique will perform the computation of approximate alignments in two pipelined phases, each phase considering the resolution of an

[1] $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$.

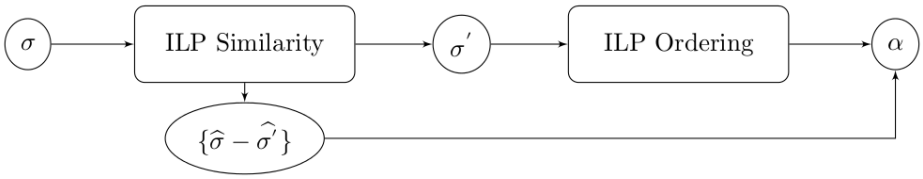


Figure 3.2: Schematic of ILP approach for computing approximate alignments.

(ILP) model containing the marking equation of the net corresponding to the model. The overall approach is described in Figure 3.2. In the first ILP model (ILP Similarity) a solution (the Parikh vector of a full firing sequence of the model) is computed that maximizes the similarity to $\widehat{\sigma}$. Elements in σ that cannot be replayed by the model in the computed Parikh vector are removed for the next ILP, resulting in the projected sequence σ'. These elements are identified as *moves in log* cf. Definition 16 in Sect. 2.2.3, and will be inserted in the approximate alignment computed α. In the second ILP model (ILP Ordering), it is guaranteed that a feasible solution containing at least the elements in σ' exists. The goal of this second ILP model is to compute the approximate alignment given a user-defined granularity: it can be computed from the finest level ($\eta = 1$), i.e., only singletons, to the most coarse level ($\eta = |\sigma|$), i.e., Parikh vector.

3.4 ILP for Similarity (Seeking an optimal Parikh vector)

Let's assume that there is a labeled Petri net N and an observed trace σ. This stage will be centered on the marking equation of the input Petri net. Let $J = \ell(\Sigma) \cap \text{supp}(\widehat{\sigma})$, i.e., the labels that appeared in the observed trace, the following ILP model computes a solution that is as similar as possible with respect to the firing of the activities appearing in the observed trace:

$$\text{Maximize } \Big(\sum_{\ell(t) \in J} X[t] - \delta \times \sum_{\ell(t) \notin J} X[t] + 0 \times \sum_{\ell(t) = \tau} X[t] \Big),$$

$$\text{Subject to:}$$

$$m_{end} = m_{start} + \mathbf{N}.X \tag{3.2}$$

$$\forall t \in X, \forall a \in \hat{\sigma} \quad \text{If} \quad \ell(t) \in J \quad \text{and} \quad \ell(t) = a : \quad \widehat{\sigma}[a] = \sum_{\ell(t) = a} (X[t] + X^s[t])$$

$$X, X^s \geq \mathbf{0}$$

δ in the objective function is a user defined value with $\delta \geq 1$, which penalizes transitions of the model which do not have any labels in J. The larger

is the value of δ, the greater penalty the elements not in J do receive. Also note that silent or invisible transitions of the model, i.e., $\ell(t) = \tau$, receive 0 cost. Hence, the model searches for a vector X that both is a solution to the marking equation and maximizes the similarity with respect to $\widehat{\sigma}$. Notice that the ILP problem has an additional set of variables $X^s \in \mathbb{N}^{|J|}$, and represents the slack variables needed when a solution for a given activity cannot equal the observed number of firings. By maximizing elements of X in J and minimizing those not in J, solutions to (3.2) clearly try to both assign zeros as much as possible to the X^s variables on the one side, and on the other side, try to do not fire the X variables not in J (i.e., activities not appearing in σ). Also if for an arbitrary event in the observed trace there are some transitions of the model with the same label then the number of firings for that event is equal to sum over all those transitions with that label.

An optimal solution X to (3.2), denoted by $\widehat{\sigma_P}$, represents the required transitions of the model and their number of occurrences which must be fired from the initial marking, m_{start}, to reach the final marking, m_{end}. Elements of $\widehat{\sigma_P}$ have the maximum similarity with respect to $\hat{\sigma}$. Obviously when σ is not a fitting trace, then $\widehat{\sigma_P} \neq \hat{\sigma}$ due to *skipped transitions*, i.e., those which are supposed to happen but did not happen, and/or *inserted transition*, i.e., those which were observed when they had not been enabled.

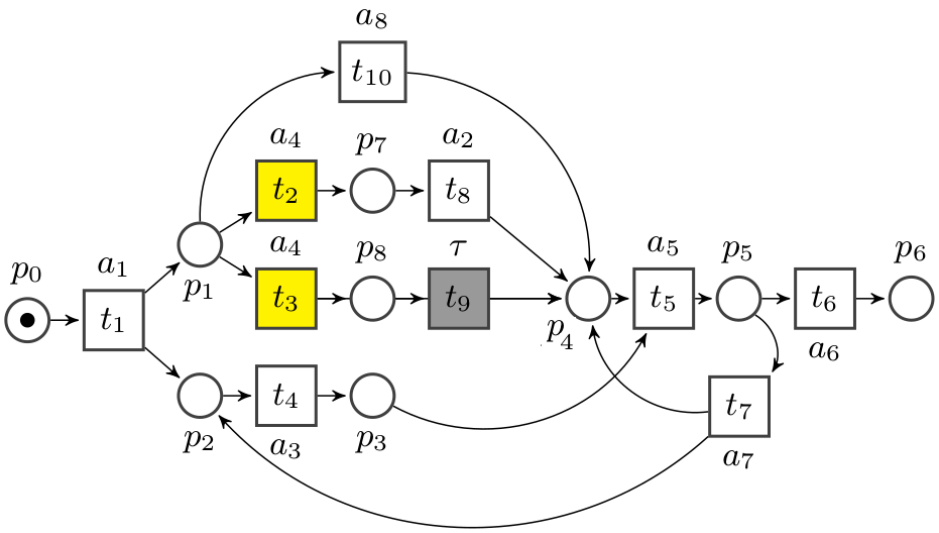


Figure 3.3: Process model

For example, consider the model depicted in Fig. 3.3 and the observed trace $\sigma = a_1a_2a_4a_5a_7a_4a_3a_6$, where $\widehat{\sigma}$ is depicted in Fig. 3.4 (b) and $J = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$. Transitions with the same label are highlighted. With the assumption of unitary costs, i.e., $\delta = 1$, by solving Eq. (3.2) for these model and trace which (described in detail in Fig. 3.5), the computed Parikh vector, X, which is called $\widehat{\sigma_P}$, and it is depicted in Fig. 3.4 (a).

Notice that a_4 happened twice in σ but based on the model only one

of them is executed since otherwise we would miss a token in place p_1. Transitions t_2 and t_3 with the label a_4 are eligible to be fired accordingly, hence in Fig. 3.5, the following constraint is proposed, i.e., $X[t_2]+X^s[t_2]+X[t_3]+X^s[t_3]=2$. Also, the occurrence of a_2 constitutes the constraint $X[t_8]+X^s[t_8]=1$. Finally, based on the just mentioned constraint, Eq. (3.2) assigns $X[t_8]=1$ and $X[t_2]=1$ to maximize the objective function and therefore increases the similarity between elements of X and $\widehat{\sigma}$, hence in Fig. 3.4 (a) it assigns $X[t_3]=0$ and $X^s[t_2]=1$ or $X^s[t_3]=1$ and $X^s[t_2]=0$ to make the corresponding constraint valid[2]. Note that a_7 occurred once in the observed trace, i.e., $\widehat{\sigma}[a_7]=1$ and therefore it suggests the constraint $X[t_7]+X^s[t_7]=1$, but t_7 will not be fired because otherwise the solution is infeasible[3], hence $X[t_7]=0$ and $X^s[t_7]=1$. Also, notice that no event corresponding to transition t_{10} of the model occurred in σ, hence in the objective function this transition was penalized. Finally, t_9 with $\ell(t_9)=\tau$ is a silent transition hence in the objective function it receives 0 cost. Finally, see Fig. 3.4 (b) which represents $\ell(X)$, i.e., the computed Parikh vector with element labels, and $\widehat{\sigma}$ to compare the similarity between elements of the two vectors.

$$X=\begin{matrix}t_1\\t_2\\t_3\\t_4\\t_5\\t_6\\t_7\\t_8\\t_9\\t_{10}\end{matrix}\begin{pmatrix}1\\1\\0\\1\\1\\1\\0\\1\\0\\0\end{pmatrix}\qquad X^s=\begin{matrix}t_1\\t_2\\t_3\\t_4\\t_5\\t_6\\t_7\\t_8\end{matrix}\begin{pmatrix}0\\1\\0\\0\\0\\0\\1\\0\end{pmatrix}$$

(a)

$$\ell(X)=\begin{matrix}a_1\\a_4\\a_4\\a_3\\a_5\\a_6\\a_7\\a_2\\\tau\\a_8\end{matrix}\begin{pmatrix}1\\1\\0\\1\\1\\1\\0\\1\\0\\0\end{pmatrix}\qquad \hat{\sigma}=\begin{matrix}a_1\\a_2\\a_3\\a_4\\a_5\\a_6\\a_7\end{matrix}\begin{pmatrix}1\\1\\1\\2\\1\\1\\1\end{pmatrix}$$

(b)

Figure 3.4: (a) ILP resolution for the model in Fig. 3.3, (b) X with labels and $\hat{\sigma}$

After having a solution of Eq. 3.2 for a given example, the observed trace σ must be updated according to the following definitions:

Definition 21 (Updating Observed Trace σ)**.** *Given an observed trace σ, X and X^s from Eq. 3.2. An event a_i will be removed from σ if, $a_i \in \hat{\sigma}, \forall t_i \in X, \ell(t_i)=a_i$ and $\sum_{t_i} X[t_i] < \hat{\sigma}[a_i]$. In case of transitions with duplicate names the total number of deletion is such that the following equation, namely $\sum_{t_i} X[t_i] = \hat{\sigma}[a_i]$ holds. However in this case all the possibilities must be considered.*

[2] Notice that the choice of assignment for X^s variables does not matter for the present approach.

[3] If t_7 is fired, then reaching the final marking based on the remaining constraints is infeasible because in that case t_5 and t_4 must be fired twice, but this contradicts the constraints $X[t_5]+X^s[t_5]=1$ and $X[t_4]+X^s[t_4]=1$.

$$\text{Maximize } (X[t_1] + X[t_2] + X[t_3] + X[t_4] + X[t_5] + X[t_6] + X[t_7] + X[t_8] - 1 \times X[t_{10}] + 0 \times X[t_9])$$

Subject to:

$$\underbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}}_{m_{end}} = \underbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{m_{start}} + \underbrace{\begin{array}{c} \\ P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \end{array} \begin{array}{c} \begin{array}{cccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix} \end{array}}_{\mathbf{N}} \times \underbrace{\begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{pmatrix}}_{X}$$

Constraints

$$\begin{aligned} &X[t_1] + X^s[t_1] = \hat{\sigma}[a_1], \\ &X[t_2] + X^s[t_2] + X[t_3] + X^s[t_3] = \hat{\sigma}[a_4], \\ &X[t_4] + X^s[t_4] = \hat{\sigma}[a_3], \\ &X[t_5] + X^s[t_5] = \hat{\sigma}[a_5], \\ &X[t_6] + X^s[t_6] = \hat{\sigma}[a_6], \\ &X[t_7] + X^s[t_7] = \hat{\sigma}[a_7], \\ &X[t_8] + X^s[t_8] = \hat{\sigma}[a_2], \\ &X \geq 0, X^s \geq 0 \\ &m_{start} \geq 0, m_{end} \geq 0 \end{aligned}$$

or

Constraints

$$\begin{aligned} &X[t_1] + X^s[t_1] = 1, \\ &X[t_2] + X^s[t_2] + X[t_3] + X^s[t_3] = 2, \\ &X[t_4] + X^s[t_4] = 1, \\ &X[t_5] + X^s[t_5] = 1, \\ &X[t_6] + X^s[t_6] = 1, \\ &X[t_7] + X^s[t_7] = 1, \\ &X[t_8] + X^s[t_8] = 1, \\ &X \geq 0, X^s \geq 0 \\ &m_{start} \geq 0, m_{end} \geq 0 \end{aligned}$$

Figure 3.5: ILP formulation for model in Fig. 3.3

In Def. 21 in the simplest case, when $X[t_i] = 0$ and $\widehat{\sigma}[a_i] > 0$, every occurrence of a_i in σ will not appear in σ'[4]. For instance in the above example, $\sum(X[t_2] + X[t_3]) = 1$, $\ell(t_2), \ell(t_3) = a_4$, and $\hat{\sigma}[a_4] = 2$, see Fig. 3.4. Thus one of a_4 in the observed trace must be removed. Therefore two updates of the given trace are $\sigma' = a_1a_2a_5a_7a_4a_3a_6$ and $\sigma' = a_1a_2a_4a_5a_7a_3a_6$.

3.5 ILP for Ordering: Computing an Aligned Step-Sequence

The schematic view of the ILP model for the ordering step is shown in Fig. 3.6. Given a granularity η, $\lambda = \lceil \frac{|\sigma'|}{\eta} \rceil$ steps are required for a step-sequence in the model that is aligned with σ'. Accordingly, the ILP model has variables $X_1 \dots X_\lambda$ with $X_i \in \mathbb{N}^{|T|}$ to encode the λ steps of the marking equation, and variables $X_1^s \dots X_\lambda^s$, with $X_i^s \in \mathbb{N}^{|J|}$ and $J = \ell(\Sigma) \cap \text{supp}(\sigma')$, to encode situations where the model cannot reproduce observed behavior in some of these steps. We now describe the ILP model in detail.

- **Objective Function**: The goal is to compute a step-sequence which resembles as much as possible to σ'. Therefore events in $\text{supp}(\sigma')$ have cost 0 in each step X_i whilst the rest have cost 1. Also, the slack variables X_i^s have cost 1.
- **Marking Equation Constraints** The computation of a model's step-sequence $m_{start} \xrightarrow{X_1} m_1 \xrightarrow{X_2} m_2 \dots m_{\lambda-1} \xrightarrow{X_\lambda} m_{end}$ is enforced by using a chain of λ connected marking equations.
- **Parikh Equality Constraints** To enforce the similarity of the Parikh vectors $X_1 \dots X_\lambda$ with respect to $\widehat{\sigma'}$, this constraints require the sum of the assignments to variables X_i and X_i^s for every variable $t \in J$ should be greater or equal to its corresponding event in $\widehat{\sigma'}[a]$, where $\ell(t) = a$. Given the cost function, solutions that minimize the assignment for the X_i^s variables are preferred.
- **Step Granularity Constraints** Require that the sum of model's steps X_i and the slack variables X_i^s is lower bounded by the given granularity η. Since the cost of variables X_i is lesser than the cost of X_i^s variables, the solutions will tend to assign as much as possible to X_i. Last step X_λ is not constrained in order to ensure the feasibility of reaching the final marking m_{end}.
- **Mimic Constraints** The input sequence σ' is split into λ consecutive chunks, i.e., $\sigma' = \sigma'_1\sigma'_2 \dots \sigma'_\lambda$, with $|\sigma'_i| = \eta$, for $1 \leq i < \lambda$. This set of constraints require at each step that the multiset of observed transitions (X_i) must only happen if it has happened in the

[4] In our experiments, only the simplest cases were encountered.

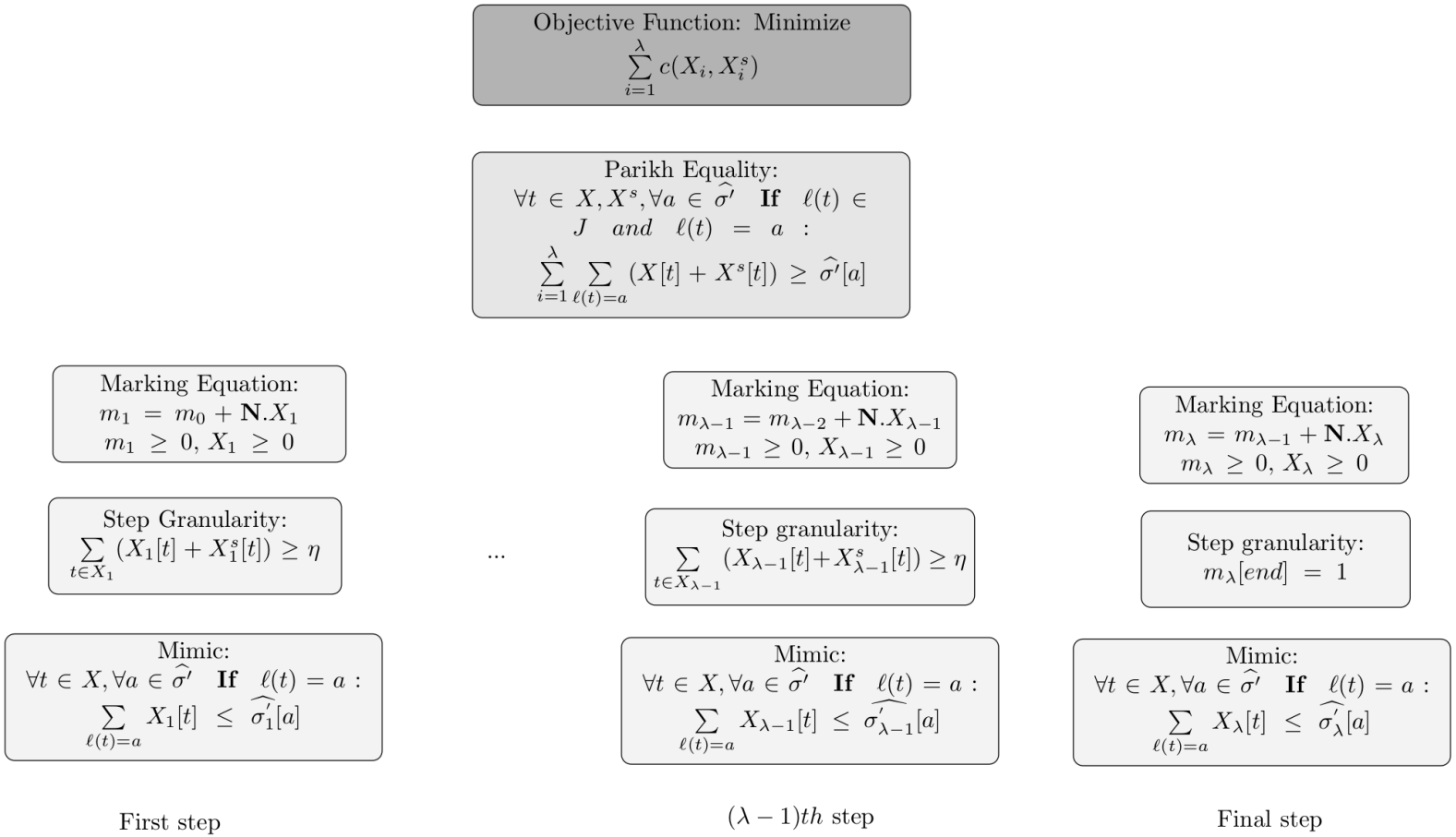


Figure 3.6: ILP schema model for the second step of Fig. 3.2

corresponding chunk σ'_i. It is worth to note that events with multiple occurrences are distinguished based on their positions.

Once the two steps of Fig. 3.2 are performed, the gathered information is sufficient to obtain an approximate alignment: on the one hand, the removed activities from the ILP model (3.2) are inserted as "moves in the log". On the other hand, the solution obtained from the ILP model of Fig. 3.6 provides the steps that can be appended to construct the final approximate alignment.

3.5.1 A note on completeness and optimality

The global optimality guarantee provided in the approach of this section is with respect to the similarity between the Parikh vectors of the computed and the observed trace. Informally, the technique searches for traces as similar as possible (c.f., ILP models (3.2)) and then computes the ordering (with respect to a given granularity). However, as the reader may have realized, by relying on the marking equation the approach presented in this section may be sensible to the existence of spurious solutions (see Subsect. 2.1.1.2). This may have negative consequences since the marking computed may not be possible in the model, and/or the Parikh vectors may not correspond to a real model trace. For the former problem (marking reachability), in case of free-choice, live, bounded and reversible nets, this problem does not exists since the structural theory completely characterizes reachability [71]. For non-structured process models (e.g., spaghetti-like) or when the Parikh vector is spurious, the technique of this chapter may still be applied, if the results obtained are verified a-posteriori by replaying the step-sequence computed. The approach presented in this chapter, will be incorporated with a divide and conquer framework in Chapter 7. The corresponding results and experiments over both well-structured and unstructured process models is showing the potentials of the technique in practice for both situations.

3.6 Outlook

This chapter presents a novel approach for alignment computation based on Integer Linear Programming (ILP) optimization technique and structural theory of Petri nets, i.e., marking equation. The presented approach, given an observed trace formulates an alignment computation in two steps. The first step considers how many of events in the observed trace can be reproduced by the model regardless of their order. The second step looks for the best ordering of the elements found already to obtain the best alignment with respect to the observed trace. Although this approach presents a new framework for alignment computation, due to the complexity of an ILP instance which is NP-complete, makes this approach inefficient to deal with medium and large instances. To tackle this issue a new computation

paradigm will be presented in Chapter 7 thus we postpone the experiments to that chapter.

Chapter 4

Incremental Integer Linear Programming

4.1 Introduction

This chapter presents an algorithm for computing alignments whose nature is in between state of the art approach [5], i.e., defined as a search for a minimal path on the product of the state space of the process model and the observed behavior, an object that is worst-case exponential with respect to the size of the model, and the technique presented in Chapter 3. In that chapter, we ground the technique on the resolution of ILP models that guides the search for solutions while constructing the derived alignment. However, the techniques of this chapter ensure the derivation of an alignment by requiring the feasibility of individual steps computed. Similar to state of the art approach in [5], the algorithm is defined on the synchronous product between the observed trace and the process model, and we use part of the ILP model (the tail of the solutions obtained at each step) as an underestimate of the cost to reach a solution. The crucial element of our approach is to incrementally construct the alignment by "jumping" over the space of solutions in a depth-first manner, using ILP models as oracles to guide the search. The approach is implemented in the open-source platform `ProM` [92], and experiments are provided which witness the distinctive capabilities of the proposed approach with respect to the state-of-the-art technique to compute alignments.

The organization of this chapter is as follows. First, the formal definition of search space of synchronous product model is given in Sect. 4.2, following that in Sect. 4.3 the mentioned search space will be explored with the use of ILP techniques. Finally, experiments and results are given in Sect. 4.4.

4.2 Search Space

According to the synchronous product Petri net defined in Sect. 2.2.4 of Chapter 2 for computing an alignment, traditional algorithms, like [5], search for alignments using a depth-first search method over a search graph in which each node represents a partial firing sequence of the system and each edge the firing of a transition. More formally the corresponding alignment's search space can be defined as follow:

Definition 22 (Search space)**.** *Let $N = \langle P, T, \mathcal{F} \rangle$ be a synchronous product Petri net where $T = T^s \cup T^l \cup T^m$ can be partitioned into sets of transitions corresponding to synchronous moves, log moves and model moves respectively and let (N, m_{start}, m_{end}) a corresponding net system. Furthermore let $c : T \to \mathbb{R}^+$ a cost function. The alignment search space is defined as $S = (V, E, c)$, with $V = \{m \mid (N, m_{start})[\sigma\rangle(N, m)\}$ and $E \subseteq V \times T \times V$ such that $(m, t, m') \in E$ if and only if $(N, m)[t\rangle(N, m')$. The root of the search space is $m_{start} \in V$ the initial marking. The target node in the search space is the final marking $m_{end} \in V$. Note that $m_{end} \in V$ since the final marking of a system net is assumed to be reachable.*

Note that, in the general case, the search space is not bounded. There may be infinitely many markings reachable from the initial marking and hence in the search space. Finding an optimal alignment is translated as finding a shortest path from m_{start} to m_{end} in the search space, where c represents the length of the edges[1].

In order to find the shortest path[2] in the search space, traditional alignment approaches use the A^* algorithm. This algorithm relies on a estimate function that underestimates the remaining costs from the current node to one of the target nodes. The cost between nodes m and m' in V can be underestimated by the marking equation, cf. Equation 2.1, but to make everything easier and derive the concepts home for the approach in this chapter, that equation will be represented as:

$$m = m_{start} - \mathbf{N}^- \cdot X + \mathbf{N}^+ \cdot X \tag{4.1}$$

Where $\mathbf{N}^-$ and $\mathbf{N}^+$ are matrices which shows consumptions and produced tokens respectively, see Fig. 2.8 in Chapter 2.

Definition 23 (Underestimating the costs)**.** *Let $S = (V, E, c)$ be a search space and $m_c \in V$ the current marking reached in the graph. We know that if there exists a σ' such that $(N, m_c)[\sigma'\rangle(N, m_{end})$ then $m_c + \mathbf{N} \cdot \widehat{\sigma'} = m_{end}$. Therefore, the solution to the linear problem minimize $c(\varsigma)$ such*

[1] Since the cost function c does not allow for 0-length, there are no loops of length 0 in the graph. In the available implementations of the alignment problem, this is hidden from the end-user when instantiating the cost function, but an $\epsilon > 0$ is used in the core computation.

[2] Note that there may be more than one shortest path. Where we talk about the shortest path, we mean any shortest path.

that $m_c + \mathbf{N} \cdot \widehat{\varsigma} = m_{end}$ provides an underestimate for the cost of σ', i.e. $c(\varsigma) \leq c(\sigma')$.

If no solution exists, the final marking cannot be reached, which implies that part of the search space is not relevant or in other words a correct underestimate for the remaining distance is infinite.

This approach to finding alignments has been implemented in ProM and has been extensively used in many applications. However, there are two problems with this approach. Firstly, the search space can be very large (although only a finite part needs to be considered). Typically, the search space size is exponential in the size of the synchronous product model which is the product of the original model and the trace to be aligned. Secondly, computing estimates is computationally expensive. This can be done both using Linear Programming and Integer Linear Programming, where the latter provides more accurate estimates. In practice however, both techniques are equally fast as the increase in precision when doing Integer computations allows the A^* algorithm to visit fewer nodes.

4.3 Search Space Exploration using ILP

4.3.1 Computing Optimal Alignments using ILP

The presented approach in this section, is very similar in nature to the technique in Chapter 3. Both approaches uses ILP to find an optimal set of Parikh vectors that reach us from initial marking to the final marking where the main difference would be that the former works with the original process model and the later takes into account the synchronous product model which is created from the process model and the observed trace.

In this chapter, we take a fundamentally different approach as we incrementally construct (possibly suboptimal) alignments. We do so, by "jumping" trough the synchronous product model in a depth-first manner until we reach the final marking. Once the final marking is reached, we terminate the search. Effectively, from a given marking, we fire a total of x transitions such that these x firings are locally optimal with respect to the cost function c and we reach the next node in the search space, from where we continue our search. However, before discussing our algorithm, we first consider a method for computing optimal alignments of a given maximal length using the marking equation.

The marking equation allows us to formalize x transition executions at once by taking the consumption matrix for each step and the marking equation for all preceding steps in the following way:

Property 1 (Marking equation for executing x transitions)**.** *Let $N = \langle P, T, \mathcal{F} \rangle$ be a Petri net, m_0, m_f two reachable markings of the net and let $\sigma = \langle t_0, \ldots, t_{x-1} \rangle$ be a trace such that $(N, m_0)[\sigma\rangle(N, m_f)$. Furthermore, for $0 < i \leq x$, let m_i be such that $(N, m_0)[\langle t_0, \ldots, t_i \rangle\rangle(N, m_i)$. Using the*

marking equation and general properties of transition firing, we know the following properties hold:

- $m_f = m_0 - \mathbf{N}^- \cdot \widehat{\sigma} + \mathbf{N}^+ \cdot \widehat{\sigma}$ *as the sequence* σ *is executable,*
- *for* $0 < i \leq x$ *holds that* $m_i = m_{i-1} - \mathbf{N}^- \cdot \widehat{\langle t_{i-1} \rangle} + \mathbf{N}^+ \cdot \widehat{\langle t_{i-1} \rangle}$*, i.e. the marking equation holds for each individual transition in the sequence,*
- *for* $0 \leq i < x$ *holds that* $m_i - \mathbf{N}^- \cdot \widehat{\sigma_{0..i}} + \mathbf{N}^+ \cdot \widehat{\sigma_{0..i-1}} \geq 0$*, i.e. before firing of each transition there are sufficient tokens to fire that transition.*

The properties above are fundamental properties of Petri nets and the marking equation [57]. They give rise to a new algorithm to find alignments of a given length.

Definition 24 (Up To Length x Alignment as ILP problem)**.** *Let* $N = \langle P, T, \mathcal{F} \rangle$ *be a synchronous product Petri net and let* (N, m_{start}, m_{end}) *a corresponding net system. Furthermore let* $c : T \to \mathbb{R}^+$ *a cost function. Let* $\theta_0, \ldots, \theta_{x-1}$ *be a set of* x *vectors*[3] *of dimension* $|T|$ *as the optimal solution to the following* $\{0, 1\}$ *ILP problem:*

$$\underset{\Sigma}{\textit{minimize}} \quad \sum_{0 \leq i < x} c(\theta_i) \tag{4.2}$$

$$\textit{subject to} \quad m_{start} + \sum_{0 \leq j < x} \mathbf{N} \cdot \theta_j = m_{end} \tag{4.3}$$

$$\forall_{0 \leq i < x} \quad {\theta_i}^T \cdot \vec{1} \leq 1 \tag{4.4}$$

$$m_{start} + \sum_{0 \leq j < i} \mathbf{N} \cdot \theta_j - \mathbf{N}^- \cdot \theta_i \geq 0 \tag{4.5}$$

$$\forall_{0 < i < x} \quad \theta_{i-1}^T \cdot \vec{1} \geq \theta_i^T \cdot \vec{1} \tag{4.6}$$

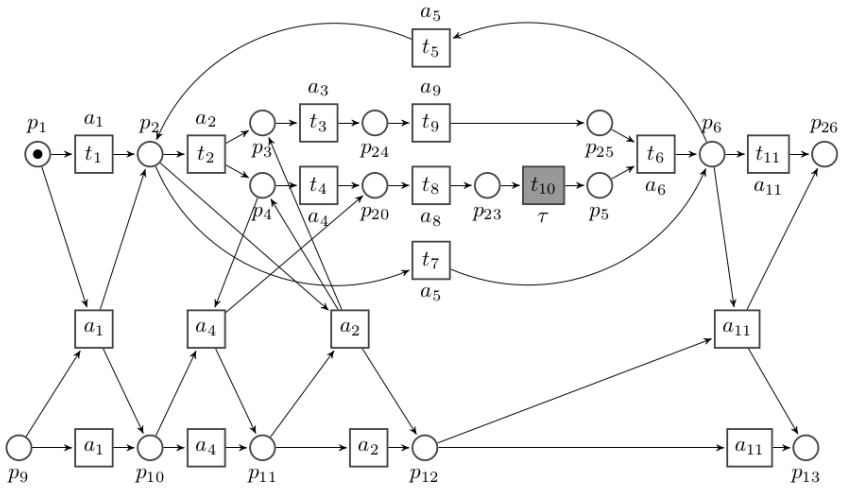
An optimal solution to the problem above constitutes a full firing sequence σ *of length* $l = \sum_{0 \leq i < x} \theta_i^T \cdot \vec{1}$ *of the net* N *in the following way: for each* $0 \leq i < l$ *holds that* $\sigma_i = t \equiv \theta(t) = 1$*, i.e. the sequence* σ *is made up of those transitions which correspond to the variables taking value* 1 *in this system. Note that for* $l \leq i < x$ *holds that* $\theta_i^T \cdot \vec{1} = 0$*.*

The target function shown as equation 4.2 above sums the costs of firing transitions in the net. Equation 4.4 ensures that each vector corresponds to at most one firing of a transition and Equation 4.5 ensures that firing all transitions t_j preceding transition t_i from the initial marking produces sufficient tokens in every place to enable transition t_i. Equation 4.6 ensures that in any solution the vectors $\theta = \vec{0}$ are grouped together and finally,

[3] Note that θ_i is a column vector and its transpose is denoted by θ_i^T.

Equation 4.3 ensures that the final marking is reached after firing at most k transitions.

To make the concepts in Def. 24 clear, consider the synchronous product model provided in Fig. 4.1 (a). The optimal solution which is a set of vectors θ_i is presented in Fig. 4.1 (b). One can see that θ_i is consist of three parts, i.e., model, synchronous and log moves with corresponding superscripts. It must be stressed that the number of steps is assumed previously, however it is difficult to have such a knowledge in reality.



(a)

$$
\theta_0 = \begin{pmatrix} t_1 & 0 \\ t_2 & 0 \\ t_3 & 0 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ \mathbf{a_1^s} & 1 \\ a_2^s & 0 \\ a_4^s & 0 \\ a_{11}^s & 0 \\ a_1^l & 0 \\ a_2^l & 0 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
\theta_1 = \begin{pmatrix} t_1 & 0 \\ \mathbf{t_2} & 1 \\ t_3 & 0 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ a_1^s & 0 \\ a_2^s & 0 \\ a_4^s & 0 \\ a_{11}^s & 0 \\ a_1^l & 0 \\ a_2^l & 0 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
\theta_2 = \begin{pmatrix} t_1 & 0 \\ t_2 & 0 \\ t_3 & 0 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ a_1^s & 0 \\ a_2^s & 0 \\ \mathbf{a_4^s} & 1 \\ a_{11}^s & 0 \\ a_1^l & 0 \\ a_2^l & 0 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
\theta_3 = \begin{pmatrix} t_1 & 0 \\ t_2 & 0 \\ t_3 & 0 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ a_1^s & 0 \\ a_2^s & 0 \\ a_4^s & 0 \\ a_{11}^s & 0 \\ a_1^l & 0 \\ \mathbf{a_2^l} & 1 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
\theta_4 = \begin{pmatrix} t_1 & 0 \\ t_2 & 0 \\ \mathbf{t_3} & 1 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ a_1^s & 0 \\ a_2^s & 0 \\ a_4^s & 0 \\ a_{11}^s & 0 \\ a_1^l & 0 \\ a_2^l & 0 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
\ldots \theta_9 = \begin{pmatrix} t_1 & 0 \\ t_2 & 0 \\ t_3 & 0 \\ t_4 & 0 \\ t_5 & 0 \\ t_6 & 0 \\ t_7 & 0 \\ t_8 & 0 \\ t_9 & 0 \\ t_{10} & 0 \\ t_{11} & 0 \\ a_1^s & 0 \\ a_2^s & 0 \\ a_4^s & 0 \\ \mathbf{a_{11}^s} & 1 \\ a_1^l & 0 \\ a_2^l & 0 \\ a_4^l & 0 \\ a_{11}^l & 0 \end{pmatrix}
$$

(b)

Figure 4.1: (a) Synchronous product model, (b) Solution of optimization problem in Def. 24

Before showing how the ILP definition above can be extended to find

alignments up to length k, we first show that any optimal alignment σ indeed corresponds to an optimal solution to this ILP for $k = |\sigma|$.

Theorem 1. *Let $N = \langle P, T, \mathcal{F} \rangle$ be a synchronous product Petri net and let (N, m_{start}, m_{end}) a corresponding net system. Furthermore let $c : T \to \mathbb{R}^+$ a cost function and σ an optimal alignment of N. We show that there is an optimal solution to the k-alignment ILP for $k \geq |\sigma|$ corresponding to σ, i.e. the ILP-alignment problem provided us with optimal alignments.*

Proof. The proof consists of two parts. First, we show that σ translates into a solution of the ILP. Then, we show that there cannot be a more optimal solution as this would imply there is a more optimal alignment.

Let $\Theta = \{\theta_0, \ldots, \theta_{|\sigma|-1}\}$ be a set of vectors, such that for all $0 \leq i < |\sigma|$ holds that $\theta_i[t] = 1$ if and only if $\sigma_i = t$, otherwise $\theta_i[t] = 0$. We show that this is a solution to the ILP of Definition 24 by enumerating the constraints:

(4.4) For all $0 \leq i < |\sigma_a|$ it trivially holds that $\theta_i^T \cdot \vec{1} = 1$,

(4.5) Since σ is a full firing sequence, we know that for each $0 \leq i < |\sigma|$ holds that $(N, m_{start})[\sigma_{0..i-1}\rangle(N, m)$ for some marking m in which transition σ_i is enabled. Furthermore, the marking equation states that $m_{start} + \mathbf{N} \cdot \widehat{\sigma_{0..i-1}} = m$ and $m - \mathbf{N}^- \cdot \widehat{\langle \sigma_i \rangle} \geq 0$.

The definition θ_i leads to the fact that $\sum_{0 \leq j < i} \theta_j = \widehat{\sigma_{0..i-1}}$, hence we conclude that $m_{start} + \mathbf{N} \cdot \sum_{0 \leq j < i} \theta_j = m$ and $m - \mathbf{N}^- \cdot \theta_i \geq 0$. Combining this yields $m_{start} + \sum_{0 \leq j < i} \mathbf{N} \cdot \theta_j - \mathbf{N}^- \cdot \theta_i \geq 0$ for all $0 \leq i < |\sigma|$,

(4.6) Since all vectors θ_i contain one element equal to 1 this is trivially true,

(4.3) Similar to the proof for Equation 4.5, this equation is satisfied.

The set of vectors Θ indeed is a solution to the ILP corresponding to the full firing sequence σ. Now we prove that no better solution to the ILP exists by contradiction. Assume there is a solution $\Theta' = \{\theta'_0, \ldots, \theta'_{|\sigma|-1}\}$ which is a solution to the ILP with a lower target function than Θ. We know we can construct a $\sigma' = \langle t_0, \ldots, t_{l-1} \rangle$ for Θ' with length $l \leq |\sigma|$ (Definition 24). Furthermore, we know σ' is a full firing sequence. Since $\sum_{0 \leq i < |\sigma'|} c(\theta'_i) < \sum_{0 \leq i < |\sigma|} c(\theta_i)$ and the relation between σ and Θ, we know that $c(\sigma') < c(\sigma)$. However, this violates the definition of σ being an optimal alignment. □

The ILP formulation above allows us to compute an optimal alignment if we know an upper bound k for the length of such an alignment. Unfortunately, such an upper bound cannot be given in advance as this would require knowledge of the alignment sought. Furthermore, the large number of variables in this ILP (the number of transitions in the synchronous product model times the length of the alignment) makes this ILP intractable in any real life setting.

4.3.2 Computing Alignments Without Optimality Guarantees

To overcome the limitations of not knowing the length of the alignment and the intractability of the ILP computation, we introduce an algorithm for incrementally computing alignments. The core idea of this algorithm, which again relies heavily on the marking equation, is the following. We use an ILP problem that constructs an exact prefix of an alignment of relatively short length (for example $x = 10$ transitions) and estimates the remainder of the alignment in the same way the A^* techniques do. Then, we execute the exact prefix of relatively small length x, compute the resulting marking and repeat the computation until we reach the target marking.

Definition 25 (k of x prefix Alignment as ILP problem)**.** *Let $N = \langle P, T, \mathcal{F} \rangle$ be a synchronous product Petri net where $T = T^s \cup T^l \cup T^m$ are the partitions of T and let (N, m_{start}, m_{end}) a corresponding net system. Furthermore let $c : T \to \mathbb{R}^+$ a cost function. We assume $k \leq |T^l|$.*

Let $\Theta = \{\theta_0, \ldots, \theta_x\}$ be a set of $x + 1$ vectors of dimension $|T|$ as the optimal solution to the following ILP problem:

$$\underset{\Sigma}{\textit{minimize}} \quad \sum_{0 \leq i \leq x} c(\theta_i) \tag{4.7}$$

$$\textit{subject to} \quad m_{start} + \sum_{0 \leq j \leq x} \mathbf{N} \cdot \theta_j = m_{end} \tag{4.8}$$

$$\sum_{t \in T^s \cup T^l} \sum_{0 \leq i < x} \theta_i[t] \geq k \tag{4.9}$$

$$\forall_{0 \leq i < x} \quad \theta_i^T \cdot \vec{1} \leq 1 \tag{4.10}$$

$$m_{start} + \sum_{0 \leq j < i} \mathbf{N} \cdot \theta_j - \mathbf{N}^- \cdot \theta_i \geq 0 \tag{4.11}$$

$$\forall_{0 < i < x} \quad \theta_{i-1}^T \cdot \vec{1} \geq \theta_i^T \cdot \vec{1} \tag{4.12}$$

$$C \cdot \theta_{x-1}^T \cdot \vec{1} \geq \theta_x^T \cdot \vec{1} \tag{4.13}$$

An optimal solution to the problem above constitutes a firing sequence σ of length $l = \sum_{0 \leq i < x} \theta_i^T \cdot \vec{1}$ of the net N identical to Definition 24. Note that the constant C in Equation 4.13 is a sufficiently large constant, for example $C = |T|^2$.

The difference between Definition 24 and Definition 25 is relatively small, but significant. The added vector θ_x in the solution does not represent a single transition execution. Instead, it represents the "tail" of the alignment, i.e. the resulting firing sequence σ is no longer a *full* firing sequence as it is not guaranteed to reach the target marking. Instead, it reaches some intermediate marking m and θ_x is a vector underestimating the cost for reaching the final marking from m identical to the underestimate function in A^* as defined in Definition 23. Once the optimal solution

to the ILP is found, the marking m reached after executing σ is taken as a new final marking and the problem is reinstantiated with that marking as initial marking.

The second important difference is the k used solely in Equation 4.9. This equation ensures that σ contains at least k transitions from the set of synchronous moves or log moves, i.e. it guarantees progress as it is a property of a synchronous product that there are no loops in the log move and synchronous move possible.

$$
\begin{array}{rc}
 & \\
t_1 & 0 \\
t_2 & 0 \\
t_3 & 0 \\
t_4 & 0 \\
t_5 & 0 \\
t_5 & 0 \\
t_7 & 0 \\
t_8 & 0 \\
t_9 & 0 \\
\theta_0 = t_{10} & 0 \\
t_{11} & 0 \\
\mathbf{a_1^s} & 1 \\
a_2^s & 0 \\
a_4^s & 0 \\
a_{11}^s & 0 \\
a_1^l & 0 \\
a_2^l & 0 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
\quad
\begin{array}{rc}
 & \\
t_1 & 0 \\
\mathbf{t_2} & 1 \\
t_3 & 0 \\
t_4 & 0 \\
t_5 & 0 \\
t_5 & 0 \\
t_7 & 0 \\
t_8 & 0 \\
t_9 & 0 \\
\theta_1 = t_{10} & 0 \\
t_{11} & 0 \\
a_1^s & 0 \\
a_2^s & 0 \\
a_4^s & 0 \\
a_{11}^s & 0 \\
a_1^l & 0 \\
a_2^l & 0 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
\quad
\begin{array}{rc}
 & \\
t_1 & 0 \\
t_2 & 0 \\
t_3 & 0 \\
t_4 & 0 \\
t_5 & 0 \\
t_5 & 0 \\
t_7 & 0 \\
t_8 & 0 \\
t_9 & 0 \\
\theta_2 = t_{10} & 0 \\
t_{11} & 0 \\
a_1^s & 0 \\
a_2^s & 0 \\
\mathbf{a_4^s} & 1 \\
a_{11}^s & 0 \\
a_1^l & 0 \\
a_2^l & 0 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
\quad
\begin{array}{rc}
 & \\
t_1 & 0 \\
t_2 & 0 \\
t_3 & 0 \\
t_4 & 0 \\
t_5 & 0 \\
t_5 & 0 \\
t_7 & 0 \\
t_8 & 0 \\
t_9 & 0 \\
\theta_3 = t_{10} & 0 \\
t_{11} & 0 \\
a_1^s & 0 \\
a_2^s & 0 \\
a_4^s & 0 \\
a_{11}^s & 0 \\
a_1^l & 0 \\
\mathbf{a_2^l} & 1 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
\quad
\begin{array}{rc}
 & \\
t_1 & 0 \\
t_2 & 0 \\
\mathbf{t_3} & 1 \\
t_4 & 0 \\
t_5 & 0 \\
t_5 & 0 \\
t_7 & 0 \\
t_8 & 0 \\
t_9 & 0 \\
\theta_4 = t_{10} & 0 \\
t_{11} & 0 \\
a_1^s & 0 \\
a_2^s & 0 \\
a_4^s & 0 \\
a_{11}^s & 0 \\
a_1^l & 0 \\
a_2^l & 0 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
\quad
\begin{array}{rc}
 & \\
t_1 & 0 \\
t_2 & 0 \\
t_3 & 0 \\
t_4 & 0 \\
t_5 & 0 \\
\mathbf{t_6} & 1 \\
t_7 & 0 \\
\mathbf{t_8} & 1 \\
\mathbf{t_9} & 1 \\
\theta_5 = \mathbf{t_{10}} & 1 \\
t_{11} & 0 \\
a_1^s & 0 \\
a_2^s & 0 \\
a_4^s & 0 \\
\mathbf{a_{11}^s} & 1 \\
a_1^l & 0 \\
a_2^l & 0 \\
a_4^l & 0 \\
a_{11}^l & 0
\end{array}
$$

Figure 4.2: Solution of optimization problem in Def. 25

To make the distinction between Definitions 24 and 25 more clear, we provide an example here. Consider the synchronous product model in Fig. 4.1 (a) and consider $x = 6$ in Def. 25. Then the solution is presented in Fig. 4.2. Note that for this example θ_5 does not show a single transitions and it represents the tail of the alignment.

Using the k of x ILP we present the sequential alignment algorithm as Algorithm 3 and using the algorithm outlined in Algorithm 3 we define an (k, x) sequential alignment.

Definition 26 ((k, x) - Sequential Alignment)**.** *Let $N = \langle P, T, \mathcal{F} \rangle$ be a synchronous product Petri net where $T = T^s \cup T^l \cup T^m$ are the partitions of T and let (N, m_{start}, m_{end}) a corresponding net system. $\sigma = Align(N, m_{start}, m_{end}, \inf, |T^l|, x, k)$ is an (k, x) sequential alignment, where $k \leq |T^l|$ and $k \leq x$.*

The sequential alignment algorithm is a recursive algorithm. It starts by solving a k of x ILP problem which for a solution is assumed to exist. After solving the ILP, the solution is compared to the previous estimate (the cost of θ_x). If the new optimal solution deviates too much from the expected solution $e' + c' \geq 2 \cdot e$ and the θ_x is non zero, i.e. the final marking

Algorithm 3 Sequential Alignment

Input: $(N, m_c, m_{end}, e, l, x, k)$ ▷ A net N, the current marking m_c, the target marking m_{end}, the last estimate for the remaining cost e, the number of events to be explained l and two parameters x and k with $k \leq x$ and $k \leq e$.
Output: A firing sequence σ
Align $(N, m_c, m_{end}, e, l, x, k)$ {
if $m_c = m_{end}$ **then**
    return $\langle\rangle$
else
    Solve $\Theta = \{\theta_0, \ldots, \theta_x\}$ as the optimal solution to the k of x ILP of
    Definition 25 and let σ be the firing sequence derived from $\theta_0 \ldots \theta_{x-1}$
    $c' = \sum_{0 \leq i < x} c(\theta_i)$
    $e' = c(\theta_x)$
    if $\theta_x \neq \vec{0} \wedge c' + e' \geq 2 \cdot e$ **then**
        return **Align**$(N, m_c, m_{end}, e, l, x + 1, min(k + 1, l))$
    else
        compute m as $m = m_c + \sum_{0 \leq i < x} \mathbf{N} \cdot \theta_i$
        $k' = \sum_{t \in T^s \cup T^l} \sum_{0 \leq i < x} \theta_i(t)$
        return $(\sigma \circ$ **Align**$(N, m, m_{end}, e', l - k', x, min(k, l)))$
    end if
end if

is not reached, then we go into a backtracking phase. We try again, with increased value of x (and k if applicable). If the initial ILP cannot be solved, i.e. no solution exist, backtracking can also be used. However, we typically assume our process models to be sound workflow models, see Def. 7 in Chapter 2.

It is easy to see that the algorithm terminates, i.e. either the final marking m_{end} is reached, or the value of x is increased until it equals the length of the shortest path from the current marking to the final marking in which case the solution of the k of x ILP becomes optimal and $\theta_x = \vec{0}$. The following lemma states this fact formally:

Lemma 1. *Let (N, m_{start}, m_{end}) and σ be in accord with Def. 26, then, x in Alg. 3 has an upper bound.*

To prove that Algorithm 3 terminates, lemma 1 must be proved. To this end, we proceed by contradiction as follows:

Proof. Given an initial marking of the process model like m_{start}, assume that the final marking m_{end} is not reachable. In other words x is infinite i.e., as long as it is increased m_{end} according to Def. 26 the final marking is not reachable. However this assumption contradicts with sound property of the process model. Stated differently a sound WF-net (see Def. 7 in Chapter 2) has option to complete property and therefore x must be finite given the observed trace. Thus after some iterations the algorithm terminates. □

4.3.3 Quality of Alignments

The sequential alignment algorithm presented in Algorithm 3 is guaranteed to terminate and to return an alignment. However, it is not guaranteed to return an optimal alignment. This is due to the fact that the marking equation used for the θ_x vector does not correspond to an actual realizable sequence. Instead, as in the original A^* approach, it merely underestimates the optimal costs to reach the final marking. As such, sub-optimal decisions may be made in each prefix. In particular, this is the case if the model contains many so-called "transition invariants", the simplest case of which are structured loops of activities.

Even if a trace perfectly fits the model, extreme cases can be devised where the sequential algorithm may construct sub-optimal alignments (although this requires the introduction of duplicate labels), while at the same time, for some classes of model and log combinations, optimality can be guaranteed. Hence, overall, it is impossible to say anything about the quality of the delivered alignment in advance. However, as the experiments in the next section show, in practical cases, the alignments are of high quality and the reduced time complexity is well worth the trade-off.

In our experiments, which we present in the next section, we considered the relative error of the costs as a measure for the quality. This relative error is defined as the cost of the sequential alignment exceeding the cost of the optimal alignment as a fraction of the cost of the optimal alignment.

4.4 Experiments

In order to assess the quality of the proposed technique, we conducted various experiments. In this section, we show one of these experiments on a real-life dataset and model. The dataset used deals with the treatment of sepsis patients in a hospital [4]. There are 1050 cases with in total 15214 events over 16 activities. There are 74 unique sequences of activities in the log and the model used contains 19 labeled transitions and 30 unlabeled routing transitions. The model is free-choice and contains both loops and parallel constructs, i.e. it belongs to the class of models considered in this chapter.

The experiments were conducted on a Core i7-4700MQ CPU with 16GB of memory, of which at most 8GB of memory were allocated to the Java virtual machine. In the interest of fairness, all algorithms were executed in single-threaded mode[5].

Figures 4.3 and 4.4 show the analysis time of aligning this log on the

[4] Sepsis Cases - Event Log. Eindhoven University of Technology. Dataset. (2016) http://dx.doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[5] The classical A^* approach can be executed in multi-threaded mode, in which case multiple traces are aligned at once. Furthermore, the Gurobi solver can also be used in multi-threaded mode, which only affects the branch-and-bound phase of the solving.

given model using three techniques, namely (1) the baseline traditional A^* [5], (2) our approach using Gurobi [32] as a backend ILP solver and (3) our approach using LpSolve [1] as a backend solver[6]. The x-axis shows the fitness of the trace (based on the baseline which guarantees optimal alignments) and for each trace, both computation time and relative error in total costs for the alignment returned are plotted. The time is plotted on the left-hand logarithmic axis and the error on the right-hand axis.

As shown in Figure 4.3, the computation time of alignments using our approach is orders of magnitude lower than when using A^*. However, in some cases, suboptimal solutions may be returned which are up to 84% off in terms of the total costs as shown in Figure 4.4. The overall error on the entire log is $7,87\%$ for Gurobi and $7,05\%$ for LpSolve. The differences between the two solvers are explained by their local decisions for optimal solutions which may lead to different choices in the alignments. For two other models in the same collection, the results are even better, with at most an 6.7% cost overestimation.

What is important to realize is that the larger errors in the cost coincide with higher computation times in the A^* implementation. Inspection of the specific cases shows that these cases suffer from the property that the estimator used in A^*, which coincides with our θ_x, performs poorly. In the A^* case, optimality is still guaranteed, but at a cost of performance, while in our approach, the "wrong" decision is made for the alignment, leading to errors.

Figure 4.3 suggests that, when cases become more fitting, the computation becomes more expensive. However, this result is misleading as the numbers are not corrected for the length of traces, i.e. the traces that are better fitting in this dataset are typically longer. Therefore, in Figure 4.5 we show the relation between the trace length and the computation time for both A^* and for our approach using Gurobi.

Figure 4.5 shows that our approach scales linearly in the length of the trace. This is expected since, for longer traces, more ILPs need to be solved. However, these ILPs are all of equal size and, since they have the same structure, of comparable complexity.

In the A^* case, we see that there is a considerably larger influence of the trace length to the time do compute alignments. The time complexity of A^* depends on two factors, namely the size of the synchronous product's statespace and the accuracy (and time complexity) of the internal heuristic used. The size of the synchronous product's statespace is the product of the model's statespace and the length of the trace, hence this also scales linearly in the trace length. The internal heuristic used in A^* is comparable to our tail computation for θ_x which, for most Petri nets, is a fairly good heuristic. As such, the performance of A^* is polynomial[7] in a linearly

[6] We did not compare our approach to [74] since the latter does not always produce a real alignment.

[7] In this case quadratic, but in general, the quality of the heuristic used in A^* degrades with

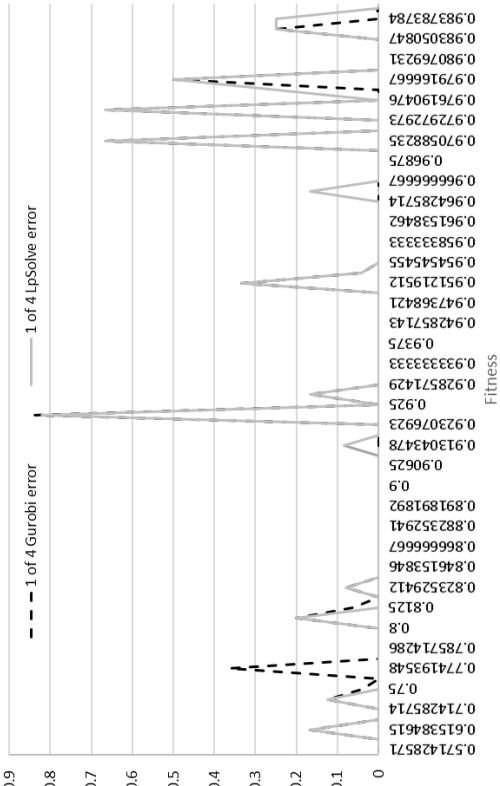


Figure 4.4: Relative error of 1-of-4 alignments.

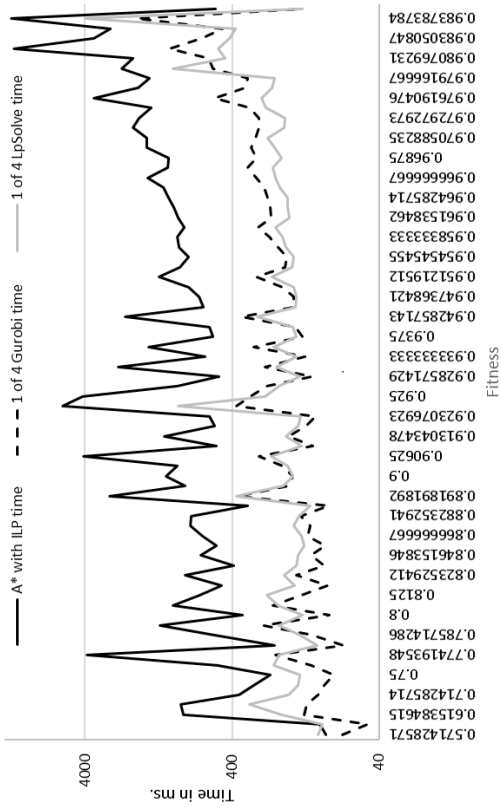


Figure 4.3: Comparison of computation times.

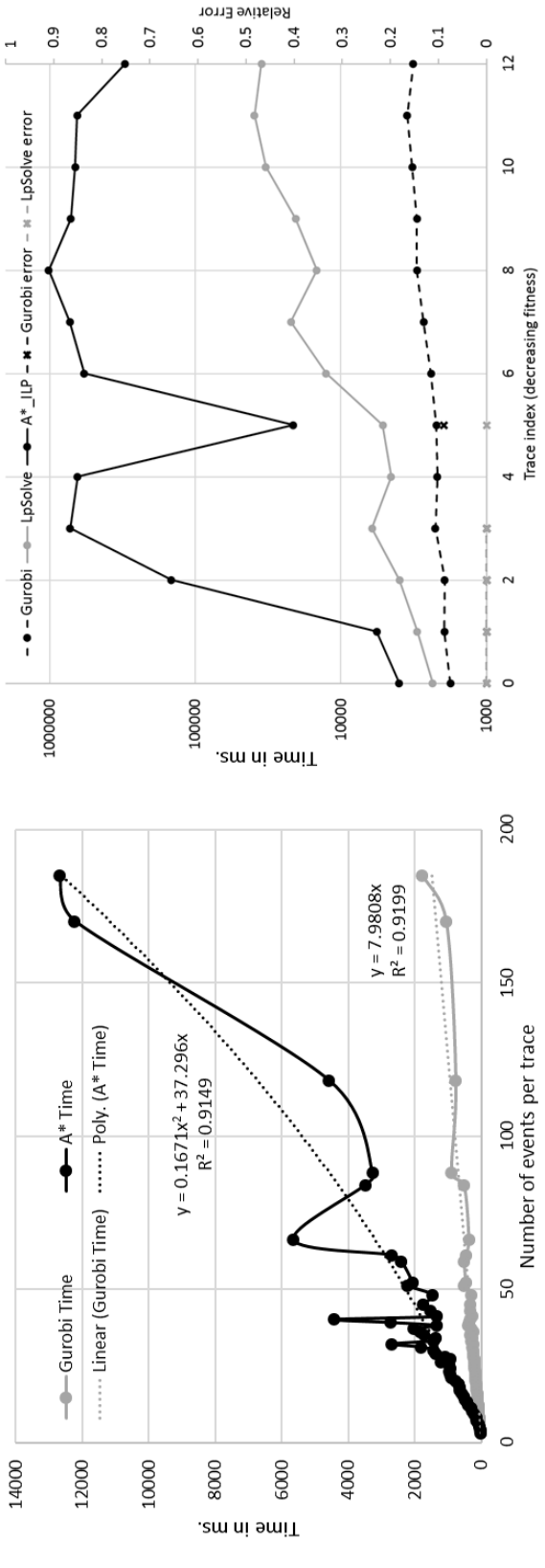


Figure 4.5: Time to compute alignments vs. length of the original trace.

Figure 4.6: Comparison of computation time and error of A^* with 1-of-4 alignments.

growing graph, which is exactly what's shown in the figure.

To emphasize the importance of our work even further, we show results on a well-known, artificial benchmark example in Figure 4.6. This example was taken from Table A.1 in Appendix A, i.e., *prFm6*, where a model is presented with 299 uniquely labeled transitions and massive parallelism. Here, we clearly see that our approach, both using LpSolve or Gurobi, can be used to find alignments for all traces within a couple of seconds. The A^* approach however, can only find alignments in some cases, before running out of time (the limit per trace was set at 200000 states, roughly corresponding to 15 minutes of computation time). Furthermore, in those cases where the A^* completes, our sequential algorithms returns optimal alignments.

In all experiments above, the cost function used was chosen in such a way that the penalties for labeling an event as a so-called log move or a transition as a so-called model move were equal to 1 and all figures were made using 1-of-4 prefix alignments. We tested various other values for both k and x and the results were comparable as long as k is significantly smaller than x. The full code is available in the anti-alignment package in ProM and is fully integrated in the conformance checking framework therein.

4.5 Outlook

This chapter presented an incremental approach to compute alignments for a given log and model using ILP as a heuristic function for A^*.

Our approach is heuristic in nature, i.e. the result is not guaranteed to be optimal, but the computation time is shown to be linear in the length of the input trace (around 8 ms per event in our experiments on a high-end laptop computer) and the error in the final results, while depending on the parameters, is shown to be reasonable.

Also, we introduced the theoretical foundations of our work, we presented the algorithm with proof of termination and we showed experimental results on real-life cases. We compared our implementation using both a freely available ILP solver as well as an industrial ILP solver with the state-of-the-art in alignment computation. Recently the work in [91] considers the heuristic function by exploiting knowledge of the traces being aligned. More specific, it uses the original trace to guarantee progress in the depth of the A^* search.

All datasets and implementations used in this chapter are freely available for download and the software is integrated in the process mining tool ProM [92].

---

the number of semi-positive transition invariants in the model, but that discussion is beyond the scope of this chapter.

Part III

Heuristic Optimization Approaches

This part of thesis adopts completely different approaches for the alignment computation problem. Both methods in Chapters 5, 6 follow iterative approaches to obtain solutions. The former, which is a form of local search, is super light and fast and the later, which is grounded on Genetic Algorithm is capable of providing multiple solutions.

Contribution: The work in Chapter 5 has been submitted as a journal paper to *ACM Transactions on Software Engineering and Methodology* [76]. The second work in this part, i.e., Chapter 6 was published in *International Conference on Business Process Management (BPM 2018)* [77]. Also those approaches have been implemented in Python as a stand-alone application framework ALI [73].

Chapter 5

Local Search Optimization Approach

5.1 Introduction

The aim of this chapter is to deploy light techniques for computing alignments for *well-formed* process models, that can be used in the large. Process models are well-formed if certain conditions on the structure of the net are satisfied. In this chapter we consider that it is better to provide an alignment, even if it is not the model trace that best resembles the observed trace (a concept defined as *optimal alignment*), than not providing an alignment at all. Dropping the optimality is not an objective of the approach presented in this chapter, but instead an artifact of the selection of techniques done, that are grounded on the structural theory of Petri nets, dynamic programming and local search methods. In practice, however, the results obtained are close to optimal.

Intuitively, the proposed technique works as follows: given a process model as a *Free-Choice Petri net* [23] satisfying the *workflow structure* and being *weakly sound* [88], and a trace, a novel replay method is proposed. It applies the structural theory of Petri nets, Sect. 2.1.1.2, to provide a set of transitions that should be fired, which is then used to select globally the set of transitions to replay the trace. To this end, first, it uses the method presented in Chapter 3, Sect. 3.4 to get an initial solution. Second, the replay relays in the workflow structure and the distance to the final marking (assumed to be a single place) to choose, among sets of enabled transitions, which one to fire at each reachable state in the replay. The replay is guaranteed to reach the final marking of the Petri net, thus providing a full sequence of the model. We then use a well-known technique from bioinformatics to align the two traces [59], and then an initial alignment is obtained. Finally, a local search technique is applied on top of the current alignment to try to improve it by merging as much as possible the deviations detected.

The techniques of this chapter have been implemented in a standalone

tool [73], and experiments done over the large problem instances publicly available nowadays reflect the capability of the technique in providing alignments in reasonable time. Remarkably, when compared to the reference technique [5], and a recent technique [67], the results are close to optimal while there is a considerably reduction both in computation time and memory consumption, often of orders of magnitude.

The chapter is organized as follows: section 5.2 provides the framework proposed in this chapter for computing alignments, which includes the three stages enumerated above. Sect. 5.3 presents the way an initial solution will be computed, this approach is partially based on materials presented in Chapter 3. Following that in Sect. 5.4 a dynamic programming approach for aligning two sequences will be presented. After that Sect. 5.5 provides a hill-climbing mechanism to improve the fitness value of the computed alignment. Finally, Sect. 5.6 reports on the set of experiments performed and shortly present the tool developed. Sect. 5.7 concludes the chapter.

5.2 Local Search Computation of Alignments

5.2.1 The Overall Perspective

Fig. 5.1 represents the overall proposed framework. The details of each part will be presented in upcoming sections. Short descriptions of each parts are presented here:

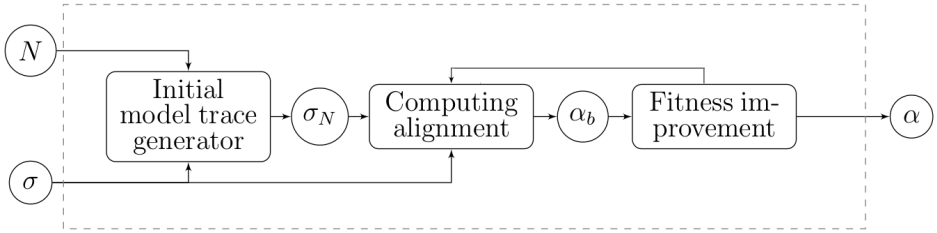


Figure 5.1: General idea for local search computation of alignments.

- **Initial modeled trace generator**. Given a model, N, and an observed trace σ, in this stage an initial modeled trace σ_N is computed. Obviously, the closer σ_N is to σ, the better. For solving this problem, some possibilities exist, based on *replay* [68, 96]. In this work we will use a novel technique that incorporates the marking equation, i.e., Eq. (2.1), in order to attain the maximum similarity to the observed trace in terms of number of events fired, a term that we denote *Parikh similarity*, see Chapter 3, Sect. 3.4. Then, since the set of firings is computed as the solution of the marking equation, a replay technique that considers the distances in the graph underlying the Petri net is used to obtain the real sequence. This approach has interesting features that makes it to avoid backtracking in the search

for a model trace. See Alg. 4 lines 2-4.

- **Computing an Alignment**. Computing an alignment between the observed and modeled traces can be done through a well-known dynamic programming approach inspired from the bioinformatics field. The result is the initial base alignment, α_b. See Alg. 4 lines 6-8.

- **Fitness improvement**. The initial modeled trace, σ_N, upon which α_b is based, is not guaranteed in general to be the best possible alignment. To improve the fitness value of the initial alignment, a local search will be done reduce the number of asynchronous moves in α_b, while preserving the executable property of the modeled trace. See Alg. 4 line 10.

This way, fitness improvement can be done iteratively, i.e., recomputing the current alignment α_b based on the previous improved alignment, until no more improvement can be made. This explains the arc-back from the fitness improvement part to the computing alignment part in Fig. 5.1.

Alg. 4 presents the general template for the framework proposed in this chapter. Notice that particular instantiations can be done in some parts of the algorithm, so that a personalized version can be obtained by selecting each important piece. For instance, by choosing one or other replay technique, a different version will be derived. Next sections will provide further details on each one of the stages.

Algorithm 4 Overall Framework

1: **Input: Global Variables** $SN(N, m_{start}, m_{end})$**,** σ ▷ Inputs are the system net, observed trace and distance matrix
2: #Initial modeled trace generator
3: $\hat{\sigma_P} \leftarrow$ Result of **Eq. 4** ▷ Computing Parikh vector, $\hat{\sigma_P}$
4: $\sigma_N \leftarrow$ **Execute** $(\hat{\sigma_P})$ ▷ Replaying $\hat{\sigma_P}$ to get model trace σ_N
5: **while** (Fitness can be improved) **do**
6: #Computing alignment
7: $M, S \leftarrow$ **Align** (σ, σ_N) ▷ Aligning σ, σ_N to get score and source matrices M, S
8: $\alpha_b \leftarrow$ **Traceback** (S) ▷ Using source matrix S, to traceback and obtain α_b
9: #Fitness improvement
10: $\alpha \leftarrow$ **Alignment-Reordering** (α_b) ▷ Reordering α_b to improve π_{α_b}(fitness value)
11: $\alpha_b \leftarrow \alpha$
12: $\sigma_N \leftarrow \alpha_b \downarrow_{A_M}$
13: **end while**
14: **Return** α_b

5.3 Initial model trace generator

Given an observed trace σ the objective of this part is to generate an executable modeled sequence σ_N, whose Parikh vector has the maximum similarity to $\widehat{\sigma}$. In this chapter we propose a new approach for this problem, which can be performed in two stages. In case of Free-choice Petri nets and weakly sound models, formal guarantees will be provided that ensure the derivation of a model trace.

The first stage, which is called ILP similarity, is originally presented in Chapter 3, Sect. 3.4. By using equation (2.1), it relies on the resolution of an ILP, which provides a Parikh vector $\widehat{\sigma_P}$ whose elements are as much similar as possible to the elements of $\widehat{\sigma}$. Given $\widehat{\sigma_P}$, the second stage is a novel technique to replay the computed Parikh vector to get an executable sequence σ_N, which excludes replaying spurious elements of $\widehat{\sigma_P}$. It will be proved that given the solution of the first stage, the second stage is complete, i.e., it is able to find a solution.

5.3.1 ILP for Similarity: Seeking for an Optimal Parikh Vector

Given a process model N and an observed trace σ, the aim of this stage as mentioned above is to get a Parikh vector called $\widehat{\sigma_P}$ with minimum distance to the Parikh vector of observed trace σ based on Eq. 3.2 in Sect. 3.4.

In short, an optimal solution X to Eq. (3.2), denoted by $\widehat{\sigma_P}$, represents the required transitions of the model and their number of occurrences which must be fired from the initial marking, m_{start}, to reach the final marking, m_{end}. Vector $\widehat{\sigma_P}$ has the maximum similarity with respect to $\widehat{\sigma}$.

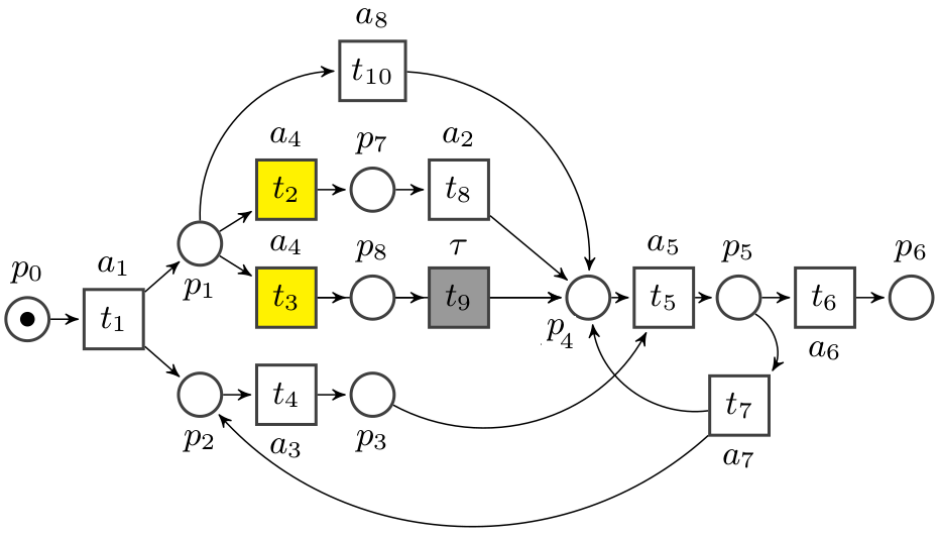


Figure 5.2: Process model N_2

As a reminder the example in Sect. 3.4 is repeated in here. Consider the model N_2 depicted in Fig. 5.2 and the observed trace $\sigma = a_1a_2a_4a_5a_7a_4a_3a_6$, where $\widehat{\sigma}$ is depicted in Fig. 5.3 (b). Solving Eq. (3.2) for these model and trace results in the Parikh vector, X, which is called $\widehat{\sigma_P}$, depicted in Fig. 5.3 (a).

$$X = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad X^s = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{matrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

(a)

$$\ell(X) = \begin{matrix} a_1 \\ a_4 \\ a_4 \\ a_3 \\ a_5 \\ a_6 \\ a_7 \\ a_2 \\ \tau \\ a_8 \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad \hat{\sigma} = \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

(b)

Figure 5.3: (a) ILP resolution for the model N_2, (b) X with labels and $\widehat{\sigma}$.

Also, see Fig. 5.3 (b) which represents $\ell(X)$, i.e., the computed Parikh vector with element labels, and $\widehat{\sigma}$ to compare the similarity between elements of the two vectors.

5.3.2 Replay The Parikh Vector: Computing An Executable Model Trace

This stage consists of executing the computed Parikh vector $\widehat{\sigma_P}$, in order to get a feasible sequence σ_N. As commented before, this step assumes a model specified over a Free-choice, weakly sound Petri net[1]. The benefits of getting a sequence, from the computed Parikh vector $\widehat{\sigma_P}$ are twofold. First, for computing the initial alignment α_b, both observed and modeled traces are needed whereas at this point there is no information about the order of elements in $\widehat{\sigma_P}$. Second, because the computed Parikh vector is the resolution of an ILP instance based on the marking equation, i.e. Eq. (2.1), it may contain some spurious elements (see section 2.1.1.2), therefore to fix this problem, i.e., getting a real solution which can be executed, $\widehat{\sigma_P}$ is replayed from the initial marking to the final marking. Obviously, there is not a unique way to execute a given Parikh vector.

The most simple way of executing a given Parikh vector $\widehat{\sigma_P}$ is based on the idea of using a tree based search approach or "generate and test", which could be done with the use of a depth-first search (DFS) mixed with a backtracking approach. This approach starts at the initial marking

[1] In spite of this assumption, in practice it performs well for more general classes, including unstructured Petri nets. This is illustrated in the experimental results provided in this chapter.

and fires transitions in $\widehat{\sigma_P}$ as much as possible, and backtracks whenever a deadlock situation is encountered. In the worst case, this approach may require to explore the full set of reachable markings in order to find a solution.

The approach proposed in this chapter prevents the backtracking mechanism. In our technique, a transition of the model is selected to be fired depending on its distance to the final marking (i.e., the place corresponding to the final marking): intuitively, a transition with the farthest distance to the final marking is on priority to be fired. To achieve this goal, the Floyd-Warshall algorithm, which computes the shortest path between two arbitrary nodes in a graph, is used as the main criteria. The computed shortest paths are represented by the distance matrix D, which is computed only once for the graph underlying the WF-net. Avoiding backtracking is the advantage of this approach, which comes at the expense of getting a modeled trace that is not guaranteed to resemble optimally the observed trace, thus potentially lowering the quality of the corresponding alignment. In practice however, the sequences obtained by this approach are rather useful in our setting. Definition 27 states the corresponding firing policy based on the mentioned criteria. It provides only the necessary criterion for a transition to be on priority of firing, whereas the sufficient condition will be discussed shortly afterwards.

Definition 27 (Firing policy)**.** *Given Parikh vector $\widehat{\sigma_P}$ and system net $SN = (N, m_{start}, m_{end})$ with $m_{end} = \{p_e\}$, where p_e is the end place in SN; Let $T_C = \{t \in T | {}^\bullet t \leq m_i\}$, i.e., enabled transitions in marking m_i, and $T_C \cap supp(\widehat{\sigma_P}) = \{t_1, t_2, ..., t_n\}$, i.e., enabled transitions that belong to $\widehat{\sigma_P}$ in marking m_i. If $D(t_k, p_e)$ represents the minimum distance of t_k to p_e then it is on priority to be fired if $\forall i \neq k : t_i \in T_C \cap supp(\widehat{\sigma_P})$, $D(t_k, p_e) > D(t_i, p_e)$.*

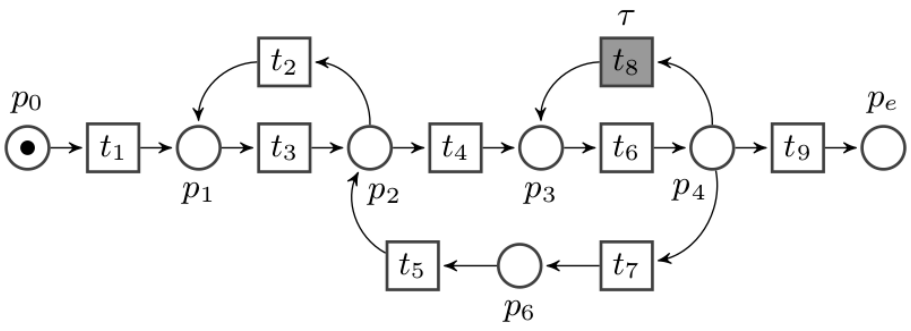


Figure 5.4: Process model N_3.

Notice that in Def. 27, there might be more than one transition in priority to be fired, in which case any of them can be fired according to the definition. Replaying $\widehat{\sigma_P}$ based on Def. 27 continues until the final marking is

reached (the proof of reaching the final marking will be presented later). After firing t_k, the marking of the model and transitions remaining in $\widehat{\sigma_P}$ are updated accordingly. Also notice that labels of transitions, i.e., duplicate or invisible labels, do not pose any problem with respect to the firing policy in Def. 27, since transitions are to be fired based on the computed Parikh vector $\widehat{\sigma_P}$ and not on $\ell(\widehat{\sigma_P})$.

Consider the model N_3 and computed Parikh vector in Figures 5.4 and 5.6 (a), respectively. The initial and final marking of the model are $m_{start}[p_0] = 1$, $m_{end}[p_e] = 1$ respectively. DFS starts at t_1, and without loss of generality let us assume that given a set of enabled transitions the one with larger subscript is chosen first, i.e, between t_8 and t_9, the latter will be expanded (fired) first. Executing $\widehat{\sigma_P}$ via backtracking by DFS, is represented in Fig. 5.6 (b) and red colors represent where the algorithm has to backtrack. On the other hand executing $\widehat{\sigma_P}$ using Def. 27 does not need any backtracking. See Fig. 5.6 (c), distances are represented on the top of each transition. The gray colored transitions represent those which are enabled at the corresponding marking, i.e. $T_C \cap supp(\widehat{\sigma_P})$, but they are not on priority to be fired due to Def. 27. For example if $m[p_2] = 1$, then both t_2 with $D(t_2, p_e) = 9$ and t_4 with $D(t_4, p_e) = 5$ are enabled, but the former will be fired since it is farther than the later from the final marking, or when $m[p_4] = 1$ then t_8 with $D(t_8, p_e) = 5$, t_9 with $D(t_9, p_e) = 1$ and t_7 with $D(t_7, p_e) = 9$ are enabled, where the later is farther with respect to others to the final marking, therefore it will be fired. Finally the computed modeled traces σ_N by two approaches are represented in Fig. 5.6 (b) and (c), respectively.

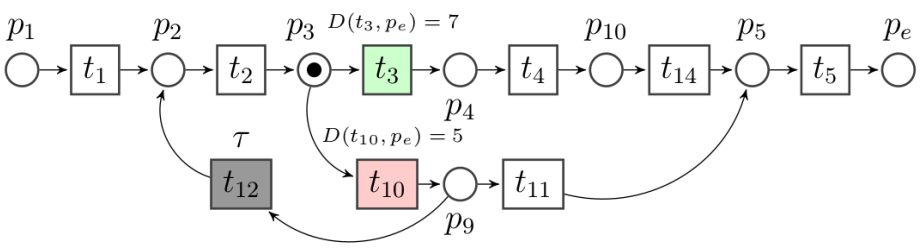


Figure 5.5: Process model N_4.

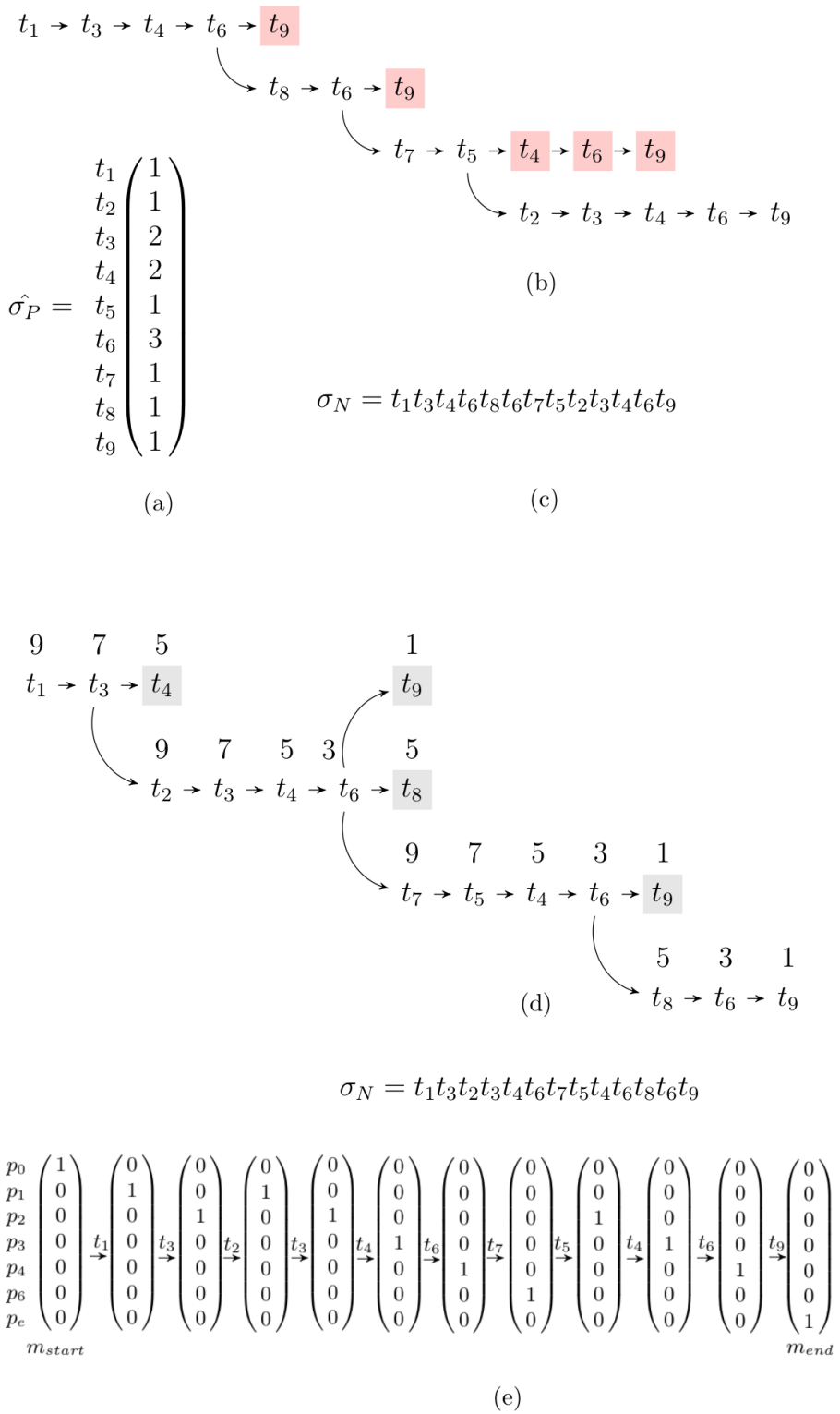


Figure 5.6: (a) Parikh vector, (b) Executing $\widehat{\sigma_P}$ using DFS, (c) The modeled trace computed by DFS, (d) Executing $\widehat{\sigma_P}$ using the proposed approach, (e) The modeled trace computed by the proposed approach

As mentioned already, replaying $\widehat{\sigma_P}$ based on Def. 27 , i.e., granting priority of firing to the farthest transition with respect to p_e avoids backtracking, but in some situations causes to miss some other transitions which are supposed to be fired based on $\widehat{\sigma_P}$. Hence, being in the farthest distance with respect to p_e, albeit necessary, is not a sufficient criterion of being on priority of firing for the technique of this chapter.

For example consider the model N_4 in Fig. 5.5, and assume that both t_3 and t_{10} are enabled in $supp(\widehat{\sigma_P})$. Based on the given marking of the model, t_3 which is the farthest transition with respect to p_e, has the priority of firing, while doing so causes t_{10} to be missed in the replay, although it is supposed to be fired according to $\widehat{\sigma_P}$. To avoid such problems, a transition with priority of firing must be fired *legitimately* according to Def. 28 as follows:

Definition 28 (Legitimacy)**.** *Assume t_k is defined according to Def. 27, and let $T_S = \{t_i | t_i \in T_C \cap supp(\widehat{\sigma_P}), {}^\bullet t_i = {}^\bullet t_k\}$, i.e., T_S represents the enabled siblings of t_k which are in $supp(\widehat{\sigma_P})$ as well, then t_k will be fired legitimately if and only if, $\exists t_j \in T_S$ and $\exists c \in \mathbb{N}$, where $D(t_k, t_j) = c$.*

Def. 28 informally states that, the priority of firing is granted to the farthest transition t_k if and only if, at least one of its siblings in T_S would be structurally accessible after it is fired. Notice that, as it happens with Def. 27, there may be more than one transition legitimated to fired according to Def. 28, which would require to pick any of them for firing. Def. 27 and 28 provide the necessary and sufficient criterion for a transition to be fired on priority which guarantees no elements in $T_C \cap supp(\widehat{\sigma_P})$ are missed, i.e. all of them will be fired without backtracking. It can be extended to all encountered reachable markings and corresponding enabled transitions of $\widehat{\sigma_P}$ given that they are not spurious (spurious elements, by definition, will not get enabled).

Theorem 2 (Completeness given legitimate firing)**.** *Given the context provided on a transition t_k by Def. 28, all elements of $T_C \cap supp(\widehat{\sigma_P})$ are fired without backtracking if and only if t_k fired legitimately.*

Prior to prove Theorem 2, first it must be established that for a reachable marking m_i, all the corresponding enabled transitions, i.e. $T_C \cap supp(\widehat{\sigma_P})$, are able to be fired in some order. Formally, it is presented in the following Lemma. This Lemma can be extended to all reachable markings from m_{start} to m_{end}.

Lemma 2. *Let Parikh vector $\widehat{\sigma_P}$[2], system net $SN = (N, m_{start}, m_{end})$ and an arbitrary marking $m_i \neq m_{end}$. $\forall t_j \in T_C \cap supp(\widehat{\sigma_P})$, there exists at least one sequence of transitions σ_O such that $supp(\widehat{\sigma_O}) \subseteq supp(\widehat{\sigma_P})$, $\widehat{\sigma_O} \leqq \widehat{\sigma_P}$ and $\widehat{\sigma_O}[t_j] = \widehat{\sigma_P}[t_j]$.*

[2] Remember that $\widehat{\sigma_P}$ is the solution of Eq. 3.2 in Chapter 3.

Proof. The proof has two parts, the first part shows the existence of $\widehat{\sigma_O}$ and the second part establishes there is a sequence of firing whose Parikh vector is $\widehat{\sigma_O}$. The second part is crucial due to the fact that Parikh vectors in general are not guaranteed to be executable (see the example provided at the end of Sect. 2.1.1.2), and reach us to the final marking from an initial marking.

- *Part 1:*
  This part proceeds by contradiction. Consider a transition like $t_j \in T_C \cap supp(\widehat{\sigma_P})$ with $\widehat{\sigma_P}[t_j] = n_j$. To proceed, separate situations are considered as follows:

  1. Let $\ell(t_j) = a_j \in supp(\widehat{\sigma})$ with $\widehat{\sigma}[a_j] = n$, in other words for transition t_j the corresponding events, i.e., a_j, happened n times in the observed trace σ. Let's assume that it is able to fire only n_i times, with $n_i < n_j$, and there is no Parikh vector to fire it n_j times. In other words there exist no $\widehat{\sigma_O}$ as stated by the lemma. But this is impossible due to violation of the following constraint of Eq. (3.2) in Sect. 3.4:

  $$\underbrace{\widehat{\sigma}[a_j]}_{n} = \underbrace{X[t_j]}_{n_j} + \underbrace{X^s[t_j]}_{n_2} + \underbrace{Res}_{n_3}{}^{3} \quad \text{where} \quad n = n_j + n_2 + n_3$$

  Stated differently, n_i times firing of t_j makes the equality sign " = " to become " > " since the difference value $n_j - n_i$ cannot be compensated by other terms in that equation. Therefore there is a Parikh vector $\widehat{\sigma_O}$ in which t_j fires n_j times.
  2. Let $\ell(t_j) = a_j \notin supp(\widehat{\sigma})$, in other words t_j is a skipped transition. Now suppose that it fires only n_i times, with $n_i < n_j$, i.e., there is no Parikh vector like $\widehat{\sigma_O}$ to fire it n_j times, and for the next markings all the enabled transitions are fired without violating any constraints with the absence of some t_j. But, this is impossible because in that case at the end we reach a solution like $\widehat{\sigma'_P}$ with $\widehat{\sigma'_P}[t_j] < \widehat{\sigma_P}[t_j]$ such that the objective function related to $\widehat{\sigma'_P}$ is greater than its counterpart, which corresponds to $\widehat{\sigma_P}$ whereas the later is supposed to be the maximum one based on given constraints in Eq. (3.2). Therefore there is a Parikh vector $\widehat{\sigma_O}$ in which t_j fires n_j times.

- *Part 2:*
  Suppose that $\widehat{\sigma_P}[t_j] = n_j > 1$, this means that t_j might be in a loop and the aim is to show that there exist a firing sequence in which t_j fires n_j times.

[3] In case of multiple transitions with the same label, for the sake of simplicity and comprehension the other corresponding terms are represented by the residual term, i.e., Res.

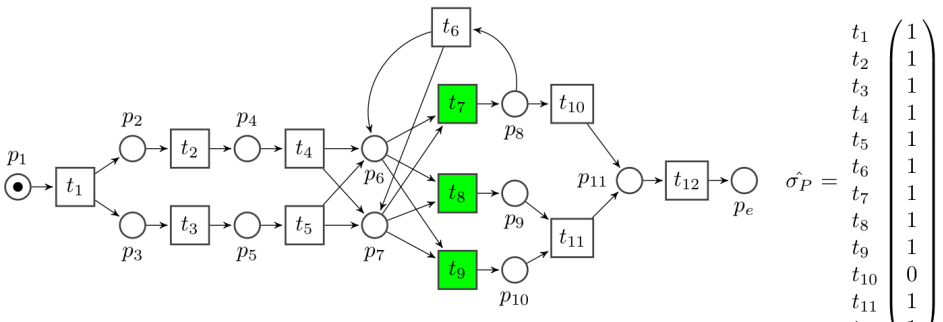
1. Lets first assume that it is in a loop. In this case given the system net, i.e., SN, one can easily form a cyclic allocation α according to Def. 10 with a non-empty domain C which contains t_j. Then based on *Cyclic Allocation Lemma* presented in [23] for live and bounded FC-models[4] there exist an occurrence sequence $m_{start} \xrightarrow{x\delta}$ such that:
   - x is a finite and contains no transitions of C
   - δ is infinite and contains only transitions allocated by α

   The mentioned Lemma states that there exist a firing sequence in which t_j fires n_j times.
2. Lets assume that t_j is not in a loop, then there should be enough tokens in ${}^\bullet t_j$ such that if fires n_j times since otherwise it violates the soundness property of the given model.

□

It must be emphasized that Lemma 2 declares, given marking m_i, for all enabled transitions of $supp(\widehat{\sigma_P})$, there exists at least one firing sequence according to their occurrences. It does not claim that all elements in $\widehat{\sigma_P}$ are able to be fired in some order entirely. Also, the following consequence can be obtained from Lemma 2.

Lemma 3. *For an arbitrary marking m_i, let $t_k \in T_C \cap supp(\widehat{\sigma_P})$ and let without loss of generality $\widehat{\sigma_P}[t_k] = n_k = 1$[5]. Then by virtue of Lemma 2, there exists a sequence of firing, σ_{T_M}, where $supp(\widehat{\sigma_{T_M}}) \subseteq supp(\widehat{\sigma_P})$, with $\widehat{\sigma_{T_M}} \leqq \widehat{\sigma_P}$ such that the following holds, $m_y = m_i + \mathbf{N} \cdot \widehat{\sigma_{T_M}}$ where ${}^\bullet t_k \leq m_y$.*



(a) (b)

Figure 5.7: (a) Process model N_5, (b) Parikh vector.

[4] We can easily make SN live by connecting the final place to the start place by a silent transition τ, a fact that does not harm the proof since the lemma holds for the elements different from τ.

[5] This corollary can be rephrased for $n_k > 1$ easily, but to keep things simple it was presented for $n_k = 1$.

Lemma 3 states that given an arbitrary marking m_i, $\forall t_k \in T_C \cap supp(\widehat{\sigma_P})$, ${}^\bullet t_k$ will get or remains marked such that t_k be able to fire n_k times. To clarify this issue, a simple example is provided. Consider the model and $\widehat{\sigma_P}$ in Fig. 5.7 (a), (b). Take a closer look at $t_7, t_8, t_9 \in supp(\widehat{\sigma_P})$ with $\widehat{\sigma_P}[t_7] = \widehat{\sigma_P}[t_8] = \widehat{\sigma_P}[t_9] = 1$ which are highlighted. Based on Lemma 2 given that they are enabled, there exists at least one firing order which contains them, for example $t_8 t_7 t_9$ or $t_7 t_9 t_8$ are two firing orders. Note that according to Lemma 3 the set of places $\{p_6, p_7\} = {}^\bullet t_7 = {}^\bullet t_8 = {}^\bullet t_9$ remains or gets marked to let t_7, t_8 and t_9 get fired according to the mentioned firing sequences. At the end of this example it must be stressed that Lemma 2 and 3 state that for an arbitrary marking there is at least one order of firing to fire all transitions in $T_C \cap supp(\widehat{\sigma_P})$ according to their occurrences.

Now based on Lemma 3, the proof of Theorem 2 is as follow:

Proof. It is assumed that the model is deadlock free (except the final marking $m_{end} = \{p_e\}$). We proceed by contradiction, namely, legitimate firing of a transition causes to miss at least one of its enabled siblings, therefore, backtracking is required. Stated differently, given the context provided by Def. 28, to fire t_j after t_k, backtracking needs to be done. But this is impossible due to the following argument. Firing t_k consumes tokens from ${}^\bullet t_k \cap {}^\bullet t_j$ that might make it unmarked which in that case based on Lemma 3 there is a set of transitions like T_M which marks ${}^\bullet t_j$. On the other hand, based on the firing policy in Def. 27 all the elements in T_M are at most as close as t_j to the final marking since otherwise based on Def. 27 they are fired earlier. Hence there is an arc-back path from T_M to elements of ${}^\bullet t_j$. By the deadlock free assumption of the model this path amounts to $D(t_k, t_j)$, thus if $D(t_k, t_j) \neq \infty$ then the places in ${}^\bullet t_j$ are marked and therefore t_j is fired without backtracking. □

In Def. 28 if t_k was unable to be fired legitimately, another candidate in T_S which fulfills Def. 28, will be selected and fired. Therefore, t_k would be the last transition to be fired for that marking. As an example, consider again the model in Fig. 5.5, based on the provided marking, t_3 and t_{10} are enabled, $D(t_3, p_e) = 7$ and $D(t_{10}, p_e) = 5$, and ${}^\bullet t_3 \cap {}^\bullet t_{10}$, but because of $D(t_3, t_{10}) = \infty$, i.e., there is no direct path from t_3 to t_{10}, the former is unable to be fired legitimately, therefore t_{10} is fired instead.

The replaying approach presented in this chapter avoids the necessity of backtracking, guarantees that the policy based on the legitimate firing will not miss the firing of any enabled transition encountered during the process. Nonetheless, for a given system net SN and $\widehat{\sigma_P}$, regardless of replaying approach the final marking m_{end} is reachable. More formally it is stated as follow:

Theorem 3 (Reachability of the final marking)**.** *Let $\widehat{\sigma_P}$ be the Parikh vector which is computed based on Eq. (3.2) in Sect. 3.4 for a given system net $SN = (N, m_{start}, m_{end})$, then $\exists \sigma_R$ such that $\widehat{\sigma_R} \leqq \widehat{\sigma_P}$ and $m_{start}[\sigma_R\rangle m_{end}$.*

Proof. Since SN is a WF-net, let ${}^\bullet m_{end} = \{t_{end}\}$ and $m^\bullet_{start} = \{t_{start}\}$, stated differently t_{start} and t_{end} are the only transitions of the model which consumes token from the initial marking and puts token to the final marking respectively. The first part of proof proceeds by contradiction. Assume that there is no σ_R by which m_{end} is reachable. Since t_{end} is the only one element which marks m_{end} this assumption indicates $t_{end} \notin supp(\widehat{\sigma_R})$ which implies $t_{end} \notin supp(\widehat{\sigma_P})$. But this is impossible since in that case the solution of Eq. (3.2) is infeasible and there is no solution. Thus t_{end} will be fired. By the same token t_{start} will be fired because it is the only one transition of the model which consumes the initial token. Put it differently $t_{start}, t_{end} \in supp(\widehat{\sigma_R})$. The second part of proof continues as follow. Due to firing of t_{end}, some elements of $\widehat{\sigma_P}$, denoted by $\widehat{\sigma_{P_k}}$[6] must be fired to mark ${}^\bullet t_{end}$ since otherwise some tokens will be missed and therefore it contradicts the solution of Eq. (3.2). Also, there is the same argument for elements of $\widehat{\sigma_{P_k}}$ as well, and it continues until we reach transition(s) of the model which consume(s) the produced token(s) by t_{start}. More formally:

$$
\begin{aligned}
m_{end} &= m_k + \mathbf{N}X_{k+1}, \qquad X_{k+1}[t_{end}] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_k \geq 0 \\
m_k &= m_{k-1} + \mathbf{N}X_k, \qquad \forall t \in \widehat{\sigma_{P_k}}, X_k[t] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_{k-1} \geq 0 \\
&\vdots \\
m_2 &= m_1 + \mathbf{N}X_2, \qquad \forall t \in \widehat{\sigma_{P_2}}, X_2[t] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_1 \geq 0 \\
m_1 &= m_{start} + \mathbf{N}X_1, \qquad X_1[t_{start}] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_{start} \geq 0
\end{aligned}
$$

This argumentation establishes the existence of a sequence like σ_R by which the final marking m_{end} is reachable from m_{start} and $\widehat{\sigma_R} \leqq \widehat{\sigma_P}$. □

It is worth stressing that the existence of a firing sequence given a Parikh vector is only guaranteed for weakly sound bounded free-choice models, and not for more general Petri net classes. A representative example is as follow, consider the the model in Fig. 5.8 which is not a FC-net and an observed trace $\sigma = a_1a_2a_8a_9a_6$, the computed Parikh vector based on Eq. 3.2, i.e., $\widehat{\sigma_P}$, contains t_1, t_2, t_8, t_9, t_6 which is not realizable in the net.

[6] If $supp(\widehat{\sigma_{P_k}}) > 1$ then these elements are fired concurrently, i.e., there is no causality among them.

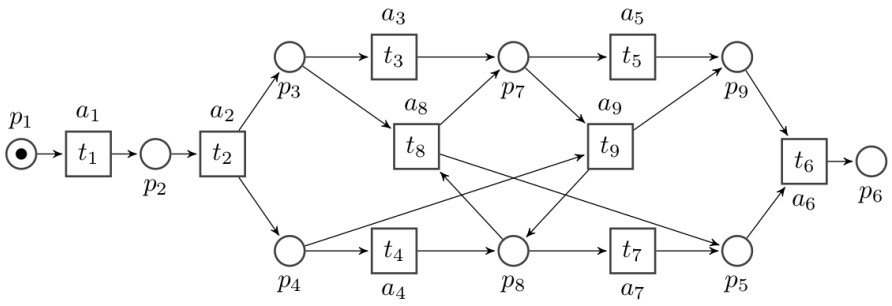


Figure 5.8: Limitation for non free-choice models.

One can see that despite of having a feasible solution for the optimization problem in Eq. 3.2, i.e., $\widehat{\sigma_P}$ (see below that all constraints are satisfied) it is unable to be fired and reach us from initial marking p_1 to final marking p_6.

$$\widehat{\sigma_P} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \widehat{\sigma} = \begin{matrix} a_1 \\ a_2 \\ a_6 \\ a_8 \\ a_9 \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, m_{end} = \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, m_{start} = \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{aligned} m_{end} &= m_{start} + \mathbf{N}.\widehat{\sigma_P}, \\ \widehat{\sigma_P}[t_1] &= \widehat{\sigma}[a_1] = 1, \\ \widehat{\sigma_P}[t_2] &= \widehat{\sigma}[a_2] = 1, \\ \widehat{\sigma_P}[t_6] &= \widehat{\sigma}[a_6] = 1, \\ \widehat{\sigma_P}[t_8] &= \widehat{\sigma}[a_8] = 1, \\ \widehat{\sigma_P}[t_9] &= \widehat{\sigma}[a_9] = 1 \end{aligned}$$

Indeed whenever places p_3 and p_4 get marked then t_8 and t_9 can fire spuriously, namely they make negative marking for p_7 and p_8 and fill in these transitions at the same time.

The mechanics of executing $\widehat{\sigma_P}$ are demonstrated by Alg. 5, 6 and 7. The global variables are: $\widehat{\sigma_P}$ and its support, which are computed as shown in the previous section, the distance matrix D and the system net, SN. Alg. 5, for a given marking, i.e., m_{curr}, identifies the farthest transition and its siblings, i.e., t_k and T_S, among the enabled set of transitions in $supp(\widehat{\sigma_P})$, line (5-7). If $T_S = \phi$ then t_k is fired simply and the current marking, m_{curr}, and $\widehat{\sigma_P}$ will be updated accordingly, line (8-10), otherwise it is examined whether it can be fired legitimately, line (12-15). If t_k was

unable to be fired legitimately, other candidates in T_S are examined in the same way until one could be fired legitimately, line (17-23). Alg. 5 continues this procedure until it reaches the final marking, hence there is a loop in line 4. Notice that silent transitions, i.e., $\ell(t) = \tau$, are removed from the computed σ_N since they are invisible.

At the end one can see that for the presented approach the distance matrix D is computed once for a given model and event log. So the time complexity is $Max\{\Theta(|P| + |T|)^3, \Theta(\|\widehat{\sigma_P}\|_1)\}$ where the first term denotes the time complexity of computing D by Floyd-Warshall algorithm which relays on the number of places and transitions in the model, and the second term represents the number of elements in the Parikh vector $\widehat{\sigma_P}$ which are supposed to be replayed.

Algorithm 5 EXECUTE $\hat{\sigma_P}$

1: **Input: Global Variables** $supp(\hat{\sigma_P})$**,** D**,** $SN(N, m_{start}, m_{end})$
2: $\sigma_N \leftarrow \phi$ ▷ Initialize the modeled trace
3: $m_{curr} \leftarrow m_{start}$
4: **while** $(supp(\hat{\sigma_P}) \neq \phi \wedge m_{curr} \neq m_{end})$ **do**
5: $T_C \leftarrow \{t \in T | {}^\bullet t \leq m_{curr}[p]\}$ ▷ Enabled transitions in the m_{curr}
6: $T_C \leftarrow T_C \cap supp(\hat{\sigma_P})$ ▷ Enabled transitions in $supp(\hat{\sigma_P})$
7: $T_S, t_k \leftarrow$ MAX_DIST(T_C) ▷ Finding the farthest transition and its siblings
8: **if** $T_S = \phi$ **then**
9: FIRE_UPDATE(t_k) ▷ Firing and updating the marking of the model
10: $\sigma_N \leftarrow \sigma_N + t_k$ ▷ Concatenating the fired transition
11: **else**
12: Paths$\leftarrow \{c \in \mathbb{N} | \exists t_j \in T_S, D(t_k, t_j) = c\}$
13: **if** Paths $\neq \phi$ **then** ▷ Examine the legitimacy definition
14: FIRE_UPDATE(t_k)
15: $\sigma_N \leftarrow \sigma_N + t_k$
16: **else** ▷ Violating the legitimacy definition
17: **while** Paths $= \phi$ **do** ▷ Looking for another transition, i.e., sibling of t_k to follow the legitimacy definition
18: $T_S \leftarrow T_S - \{t_k\}$
19: $T_S, t_k \leftarrow$ MAX_DIST(T_S)
20: Paths$\leftarrow \{c \in \mathbb{N} | \exists t_j \in T_S, D(t_k, t_j) = c\}$
21: **end while**
22: FIRE_UPDATE(t_k)
23: $\sigma_N \leftarrow \sigma_N + t_k$
24: **end if**
25: **end if**
26: **end while**
27: **Return** σ_N

Algorithm 6 MAX_DIST

1: **Input:** T_{in} ▷ T_{in}, is a set of transitions
2: $t_k \leftarrow \{t \in T_{in} | \forall t_i \in T_{in}, D(t) > D(t_i)\}$ ▷ Farthest transition, i.e, t_k
3: $T_S \leftarrow \{t_i \in T_{in} | {}^\bullet t_i =^\bullet t_k\}$ ▷ Siblings of a_{max}
4: **Return** T_S, t_k

Algorithm 7 FIRE_UPDATE

1: **Input:** t_k
2: Initialize X with 0, $X[t_k] \leftarrow 1$
3: $m_{next} = m_{curr} + \mathbf{N} \cdot X$
4: $\hat{\sigma_P}[t_k] \leftarrow \hat{\sigma_P}[t_k] - 1$ ▷ Updating $\hat{\sigma_P}$
5: Update $supp(\hat{\sigma_P})$
6: $m_{curr} \leftarrow m_{next}$ ▷ Updating the current marking
7: **Return**

5.3.3 The feasibility of executing $\widehat{\sigma_P}$

It is worth to point out that in the procedures just mentioned for executing $\widehat{\sigma_P}$ to obtain σ_N, $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$. This is due to the fact that computing $\widehat{\sigma_P}$ by the formulation in Eq. (3.2) relays on marking equation, i.e., Eq. (2.1), therefore on occasions it may have spurious elements, i.e., those which are not reachable during replaying. Therefore in the worst case no complete modeled trace σ_N with the following condition i.e., $\widehat{\sigma_N} = \widehat{\sigma_P}$, is guaranteed to be generated neither by the proposed approach nor other replaying techniques. But based on the following Theorem, the proposed approach always finds a sequence like σ_N such that $m_{start}[\sigma_N\rangle m_{end}$.

Theorem 4 (Existence of the modeled trace)**.** *For the proposed replaying approach given the contexts of Def. 27, 28,* $\exists\sigma_N$ *where* $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$ *such that* $m_{start}[\sigma_N\rangle m_{end}$.

Proof. The proof proceeds by contradiction. Suppose that for the proposed approach $\nexists\sigma_N$ such that $m_{start}[\sigma_N\rangle m_{end}$. As such, it means the proposed technique reaches to an arbitrary marking $m_i \neq m_{end}$ for which there are no enabled transitions, i.e., $T_C = \phi$. But this is impossible due to the following reasons:

1. There is no deadlock in the model (except the final marking)
2. By virtue of Theorem 3, $\exists\sigma_R$ such that $m_{start}[\sigma_R\rangle m_{end}$ therefore the initial assumption made at the beginning of the proof implies backtracking since $m_i \geq 0$ and $m_i \neq m_{end}$ but it contradicts the presented replaying technique since by virtue of Theorem 2 it does not backtrack.

Therefore the presented technique finds a sequence like σ_N such that $m_{start}[\sigma_N\rangle m_{end}$. □

Theorem 4 states that the proposed technique finds a sequence like σ_N by which m_{end} is reachable from m_{start}, in other words the technique is complete. The following theorem proves that σ_N is the longest sequence among the existing sequences by which the final marking is reachable from the initial marking.

Theorem 5 (Length optimality of modeled trace)**.** *Given the context of Theorem 4,* $\nexists\sigma'_N$ *where* $\widehat{\sigma_N} \leqq \widehat{\sigma'_N} \leqq \widehat{\sigma_P}$ *and* $|\sigma_N| < |\sigma'_N|$ *such that* $m_{start}[\sigma'_N\rangle m_{end}$.

Proof. We present the proof by contradiction as follows. Suppose that $\exists\sigma'_N$ where $\widehat{\sigma_N} \leqq \widehat{\sigma'_N} \leqq \widehat{\sigma_P}$ and $|\sigma_N| < |\sigma'_N|$ such that $m_{start}[\sigma'_N\rangle m_{end}$, in other words $\exists t_i \in supp(\widehat{\sigma'_N})$ but $t_i \notin supp(\widehat{\sigma_N})$. The mentioned assumption implies that there is marking m'_i such that ${}^\bullet t_i \leq m'_i$ and it is not met while σ_N is being obtained. But this is impossible to have such an intermediate marking for the following reason. If we consider the corresponding reachability markings for two sequences, i.e., σ_N and σ'_N, as follows:

$$\begin{aligned} &m_{start} \xrightarrow{t_{start}} \cdots m_i \cdots \xrightarrow{t_{end}} m_{end}, && \sigma_N \\ &m_{start} \xrightarrow{t_{start}} \cdots m_i \xrightarrow{t_j} m'_i \xrightarrow{t_i} m'_{i+1} \cdots \xrightarrow{t_{end}} m_{end}, && \sigma'_N \end{aligned}$$

Then to reach m'_i some transitions like t_j[7] from the previous marking, lets say m_i, must be fired. If m_i is reachable while σ_N is being replied then m'_i is also reachable by virtue of Theorem. 2, since all the enabled transitions of m_i will be fired. Indeed it is impossible to have such a marking like m'_i and corresponding previous markings like m_i which are not met while σ_N is replayed since the initial marking of both σ_N and σ'_N is m_{start}.

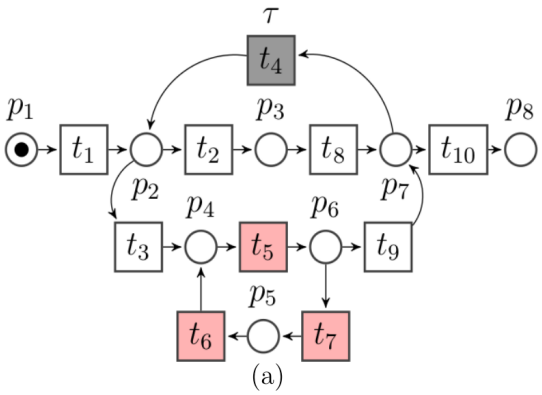
□

Since $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$ then some elements of $\widehat{\sigma_P}$ are spurious, i.e., never get enabled, and hence will not be fired. This is regardless of having well-formed or not WF-net models, but for unstructured models, i.e., *Spaghetti*, $\widehat{\sigma_P}$ may have in general more spurious elements. To enlighten this issue consider the model N_6 in Fig. 5.9 (a) and observed trace $\sigma = t_1t_2t_5t_7t_6t_8t_{10}$. The computed Parikh vector $\widehat{\sigma_P}$ is represented in Fig. 5.9 (b). One can see that $\widehat{\sigma_P}[t_5] = \widehat{\sigma_P}[t_6] = \widehat{\sigma_P}[t_7] = 1$ whereas the corresponding counterparts in $\widehat{\sigma_N}$ are zero. This is because those elements are spurious and never get enabled while $\widehat{\sigma_P}$ is being replayed; also note that they do not violate constraints of Eq. (3.2) while $\widehat{\sigma_P}$ is computed.

5.4 Aligning σ and σ_N

This section is centered around aligning a model trace σ_N, computed in the previous section, and an observed trace σ. The computed alignment

[7] To make everything simple assumes only one transition is enough. It can be easily rephrased for the case with many transitions.



(a)

$$\hat{\sigma} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad \hat{\sigma_P} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad \hat{\sigma_N} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\sigma_N = t_1 t_2 t_8 t_{10}$$

(b)

Figure 5.9: (a) Process model N_6, (b) Related Parikh vectors.

is called initial or base alignment, α_b. It is called initial, since its fitness value might be improved in the next stage.

Since both σ_N and σ can be seen as strings, we can adopt string matching algorithms to compute α_b. The string or sequence similarity problem is a well known problem in computer science, that falls under the class of algorithms computed efficiently through *dynamic programming*. Giving two arbitrary sequences, instead of determining the similarity between sequences as a whole, dynamic programming tries to build up the solution by determining all similarities between arbitrary prefixes of the two sequences. The algorithm starts with shorter prefixes and uses previously computed results to solve the problem for larger prefixes. To this end, Needleman-Wunsch algorithm [59], attempts to maximize similarity between the two input sequences by employing a scoring matrix to penalize gaps and mismatches among them. The scoring matrix can be obtained by aligning the corresponding two sequences. Formally, the alignment of two *sequences* defined by [59] in the organized way is as follow [58].

Definition 29 (Alignment of Sequences)**.** *Assume that* $\mathbf{S}$ *represent the alphabet and let* S_A *and* S_B *be two members of* $\mathbf{S}^+$ *with length* m *and* n

respectively. Let $\perp$ *denote the empty set, then:*

- (A, B) *is a match, if* $A \in S_A$, $B \in S_B$ *and* $A = B$
- (A, B) *is a mismatch, if* $A \in S_A$, $B \in S_B$ *and* $A \neq B$
- (A, B) *is a gap, if* $A = \perp$ *and* $B \in S_B$
- (A, B) *is a gap, if* $A \in S_A$ *and* $B = \perp$

If the set of moves is shown by S_M *then given an alignment* $\alpha \in S_M^*$, *the projection of the first element (ignoring* $\perp$*) results in* S_A, *and projecting the second element (ignoring* $\perp$*) results in* S_B.

In Def. 29, match and gap are the same as synchronous and asynchronous moves in Def. 16, but mismatch has no counterpart in the former. So after aligning modeled and observed trace, i.e., σ_N and σ, by this method, if a mismatch pair (X, Y) with $X \in \sigma, Y \in \sigma_N$ and $\ell(Y) \neq X$ occurs, then it will be transformed to $(X, \perp)$ and $(\perp, Y)$ to preserve properties of Def.16.

To this end, a primary matrix is created, where the first row and column are filled by observed and modeled traces respectively, as depicted in Fig. 5.10 (a). The second row and the second column are initialized with numbers starting from 0,-1,-2,..., they are depicted in yellow color in the figure. The task then is to fill the remaining cells with the recurrence Eq. (5.1), in which δ represents the gap penalty, and $s(t_i, a_j)$ represents both the match and mismatch cost between two elements t_i and a_j which are modeled and observed trace elements, respectively.

$$SIM(t_i, a_j) = \text{MAX} \begin{cases} SIM(t_{i-1}, a_{j-1}) + s(t_i, a_j) \\ SIM(t_{i-1}, a_j) - \delta \\ SIM(t_i, a_{j-1}) - \delta \end{cases} \qquad s(t_i, a_j) = \begin{cases} \beta & \text{If} \quad \ell(t_i) = a_j \\ -\beta & \text{If} \quad \ell(t_i) \neq a_j \end{cases} \tag{5.1}$$

To enlighten of how to fill the scoring matrix, consider $\sigma_N = t_1 t_4 t_5 t_6$ with $\ell(t_1) = a_1, \ell(t_4) = a_4, \ell(t_5) = a_5, \ell(t_6) = a_6$ and observed trace $\sigma = a_1 a_4 a_6$. Also assume $\delta = 1$ and $\beta = 1$. Generally speaking, by the approach of [59] filling the primary matrix is as follows: start from the up left corner as depicted in Fig. 5.10 (b). Move through the cells row by row, calculating the score for each cell by Eq. (5.1). The score is calculated as the best possible score (i.e. highest) from existing scores to the left, top or top-left (diagonal). When a score is calculated from the top, or from the left this represents move in model and move in log in the alignment respectively, see Fig. 5.10 (c). When the score is calculated from the diagonal, this might represent a synchronous move between two elements in the final alignment [8]. The final scoring matrix is depicted in Fig. 5.10 (d). Once

[8] In some situation as mentioned in the context, alignment (A, B) might represent a mismatch, hence in this case it will be decomposed to $(A, \perp)$ and $(\perp, B)$ to obey Def. 16.

it is computed, given δ and β, the bottom right entry of the matrix gives the maximum score, i.e., the number which indicates maximum similarity among the given strings, among all possible alignments.

(a)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1			
t_4	-2			
t_5	-3			
t_6	-4			

(b)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1		
t_4	-2			
t_5	-3			
t_6	-4			

$$s(t_1, a_1) = 1$$
$$SIM(t_1, a_1) = MAX \begin{cases} SIM(0,0) + s(t_1, a_1) = 1 \\ SIM(t_1, 0) - 1 = -2 \\ SIM(0, a_1) - 1 = -2 \end{cases}$$

(c)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	
t_4	-2			
t_5	-3			
t_6	-4			

$$s(t_1, a_4) = -1$$
$$SIM(t_1, a_4) = MAX \begin{cases} SIM(0, a_1) + s(t_1, a_4) = -2 \\ SIM(t_1, a_1) - 1 = 0 \\ SIM(0, a_4) - 1 = -3 \end{cases}$$

(d)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	-1
t_4	-2	0	2	1
t_5	-3	-1	1	0
t_6	-4	-2	0	2

$$s(t_6, a_6) = 1$$
$$SIM(t_6, a_6) = MAX \begin{cases} SIM(t_5, a_4) + s(t_6, a_6) = 2 \\ SIM(t_6, a_4) - 1 = -1 \\ SIM(t_5, a_6) - 1 = -1 \end{cases}$$

Figure 5.10: (a) Primary matrix, (b) and (c) Filling the matrix, (d) Final scoring matrix

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	-1
t_4	-2	0	2	1
t_5	-3	-1	1	0
t_6	-4	-2	0	2

(a)

α_b=

a_1	a_4	$\perp$	a_6
t_1	t_4	t_5	t_6

(b)

Figure 5.11: (a) Trace back of the scoring matrix, (b) The computed alignment

To compute the alignment that actually gives this score, we start from the bottom right entry, and compare the value with three possible sources, i.e., top, left and diagonal to identify from which one of them it came from. Obviously if it was fed by a diagonal entry, it represents a synchronous moves between corresponding elements. If it was fed by a top or left entries, then it represents an asynchronous move or a gap, see Fig. 5.11 (a). Following the above described steps, the alignment of two sequences can be found which represented in Fig. 5.11 (b). Note that given δ and β, the score of α_b is (+1)+(+1)+(-1)+(+1)=2. Also based on Def. 18 in Chapter 2, Sect. 2.2.3, by assuming $\delta_S = \delta_L = \delta_M = 1$, α_b has the fitness value, 1-1/4 or 75%. The important point to be noted here is that in some situations there may be two or more possible alignments between the two sequences which has the same maximum score, i.e., α_b is not unique.

The corresponding procedures for computing α_b is represented in Alg. 8 and 9. Alg. 8 starts to compute the scoring matrix M for given σ_N and σ by initializing it, lines 4-9. Then, it fills each entry according to Eq. (5.1), lines 11-21, and at the same time for each entry books the source of the computed score, i.e., *top, left or diagonal*, in matrix S, line 21. Alg. 9 to obtain α_b uses the source matrix S to trace back by starting at its right bottom element $S[|\sigma_N|, |\sigma|]$, lines 3-4, to find a path to the top left element $S[1, 1]$, lines 5-16. According to the content of the element under consideration, the corresponding synchronous or asynchronous moves are added to α_b. This procedure continues until it reaches the top left element. The time complexity of computing α_b is related to the computation of scoring matrix M, which is $\Theta(|\sigma_N| * |\sigma|)$.

Algorithm 8 Align σ, σ_N

Input: σ, σ_N, ℓ, β, δ ▷ ℓ is the labeling functions of transitions in σ_N
$M \leftarrow \phi$ ▷ M is the primary scoring matrix with dimension $(|\sigma_N|+1)*(|\sigma|+1)$
$S \leftarrow \phi$ ▷ S is the source matrix with dimension $(|\sigma_N|)*(|\sigma|)$ which books the source of scores
for $i \leftarrow 0$ to $|\sigma|$ **do** ▷ Initializing the first row of M
    $M[0,i] \leftarrow -i$
end for
for $j \leftarrow 0$ to $|\sigma_N|$ **do** ▷ Initializing the first column of M
    $M[j,0] \leftarrow -j$
end for
---
for $j \leftarrow 1$ to $|\sigma_N|$ **do** ▷ Filling the scoring matrix M
    for $i \leftarrow 1$ to $|\sigma|$ **do**
        if $(\ell(\sigma_N[j]) = \sigma[i])$ **then** ▷ Match between elements
            $s(\sigma_N[j], \sigma[i]) \leftarrow \beta$
        else ▷ Mismatch between elements
            $s(\sigma_N[j], \sigma[i]) \leftarrow -\beta$
        end if
        diag $\leftarrow M[j-1, i-1] + s(\sigma_N[j], \sigma[i])$
        top $\leftarrow M[j-1, i] - \delta$
        left $\leftarrow M[j, i-1] - \delta$
        $M[j,i], S[j,i] =$Max(diag, top, left) ▷ Computing the maximum score and the name of the source, i.e., diag, left, top
    end for
end for
Return M, S

Algorithm 9 Traceback

Input: S ▷ S is the source matrix computed by previous algorithm
$\alpha_b \leftarrow \phi$
$j \leftarrow |\sigma_N|$ ▷ Trace back from the bottom right element of S
$i \leftarrow |\sigma|$
while $(i \neq 0 \vee j \neq 0)$ **do**
    if $(S[j,i] =$diag) **then** $\alpha_b \leftarrow \alpha_b + (\sigma[i], \sigma_N[j])$ ▷ Synchronous move
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    end if
    if $(S[j,i] =$top) **then** $\alpha_b \leftarrow \alpha_b + (\perp, \sigma_N[j])$ ▷ Move in model
        $j \leftarrow j - 1$
    end if
    if $(S[j,i] =$left) **then** $\alpha_b \leftarrow \alpha_b + (\sigma[i], \perp)$ ▷ Move in log
        $i \leftarrow i - 1$
    end if
end while
Return α_b

5.5 Fitness improvement by local search

Having aligned σ and σ_N only provides the optimal alignment between these two sequences and it is not guaranteed that it is the best one by which the model mimics the observed trace. The root cause of this problem is not difficult to grasp and as mentioned, comes from the fact that there may be several traces which can be obtained by executing the computed Parikh vector $\widehat{\sigma_P}$. Therefore the initial computed alignment α_b in Sect. 5.4, might be far away from what is desired, i.e., the optimal one.

This section is centered around of *reordering* the initial alignment α_b, with the aim of improving the corresponding fitness value. This is obtained by trying to increase the number of synchronous moves between observed and modeled traces, given that the modeled trace σ_N remains executable. More rigorously, the reordering of α_b can be accomplished by a local search approach in which a move in log, i.e. $(a_i, \perp)$, will be merged by the corresponding move in model, i.e., $(\perp, t_j)$ and $\ell(t_j) = a_i$, to make a synchronous move, i.e., (a_i, t_j). This can only be done if: first, the resulted modeled trace remains executable, and second, the order of events in the observed trace remains unchanged. The iterative approach stops once no more merging can be made. To clear up this idea, first the formal definitions will be presented and then an illustrative example will be provided.

Definition 30 (Alignment reordering)**.** *Given a system net* $SN(N, m_{start}, m_{end})$ *and an alignment* α*. Let* $\sigma_\perp$ *and* $\sigma_{N,\perp}$ *be the projection of* α *onto the first and second elements (including* $\perp$*), respectively. The move in log* $(a_i, \perp)$ *with* $a_i = \sigma_\perp[i]$*, can be merged with the move in model* $(\perp, t_j)$ *with* $t_j = \sigma_{N,\perp}[j]$*, to make the synchronous move* (a_i, t_j) *given that the following conditions are met:*

- $\ell(t_j) = a_i$
- *After merging, the new modeled trace* $\sigma'_{N,\perp}$ *with the following changes:*

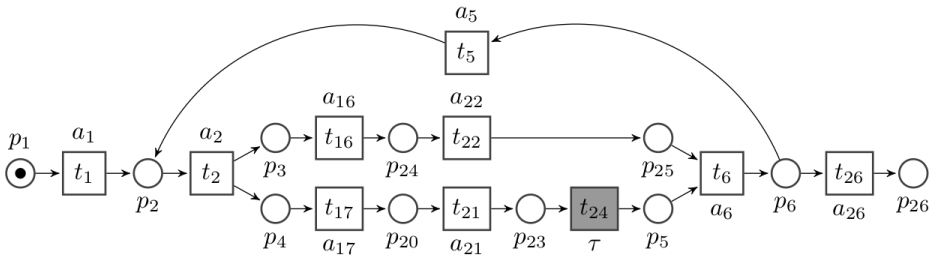
$$\sigma'_{N,\perp}[k] = \begin{cases} \sigma_{N,\perp}[k] & \text{if } k \neq i, j \\ \perp & \text{if } k = j \\ t_j & \text{if } k = i \end{cases}$$

must be executable, i.e., $(N, m_{start})[\sigma'_{N,\perp}\rangle(N, m_{end})$.

Def. 30 states that, the reordering of an alignment like α via merging the corresponding moves in log and model, i.e., increasing number of synchronous moves by flipping corresponding elements in $\sigma_{N,\perp}$, results in improving the fitness value of α, i.e., π_α. Stated differently the merging is a means by which the number of asynchronous moves in α are decreased. As implicitly stated in Def. 30, it must be emphasized that, after merging, the order of events in the given observed trace, i.e., σ, will not be changed.

Based on Def. 30, the reordering of the initial alignment α_b is done with the aim of increasing the number of synchronous moves between modeled

and observed traces. It can be done in an iterative way as long as no more synchronous matching can be obtained, hence the arc back from fitness improvement part to computing alignment part in Fig. 5.1.



(a)

$\alpha_b =$

a_1	a_2	a_{16}	$\perp$	a_{22}	$\perp$	a_{21}	a_{17}	$\perp$	a_6	a_{16}
t_1	t_2	$\perp$	t_{17}	$\perp$	t_{16}	t_{21}	$\perp$	t_{22}	t_6	t_{26}

(b)

$\alpha_b =$

a_1	a_2	a_{16}	$\perp$	a_{22}	a_{21}	a_{17}	a_6	a_{16}
t_1	t_2	t_{16}	t_{17}	t_{22}	t_{21}	$\perp$	t_6	t_{26}

(c)

Figure 5.12: (a) Process model N_7, (b) Initial alignment α_b, (c) Alignment after reordering

To illustrate how the reordering of an alignment works, a simple example is presented. Consider the model and initial alignment α_b in Fig. 5.12 (a), (b) respectively. The appropriate asynchronous moves which are candidate to merge and create corresponding synchronous moves are highlighted with the same color in Fig. 5.12 (b). One can see that $(a_{16}, \perp)$ and $(\perp, t_{16})$, highlighted in green, can be merged to make the synchronous move (a_{16}, t_{16}) and the resulted modeled trace remains executable. There is the same story for moves $(\perp, t_{22})$, $(a_{22}, \perp)$, highlighted in blue, to make (a_{22}, t_{22}). Notice that two synchronous moves $(\perp, t_{17})$, $(a_{17}, \perp)$ can not be merged since otherwise the resulted modeled trace is not executable, put it differently, in that case, t_{21} will be fired before t_{17} which causes to lose one token in p_{20}.

5.5.1 A large example

This subsection presents a large example, to shed light on how the fitness value of and initial alignment, α_b, can be significantly by the reordering technique in Def. 30 just mentioned in the previous section.

Consider the model in Fig. 5.13 (a), observed and modeled traces σ, σ_N in Fig. 5.13 (b). σ_N was computed by the method in Sect. 5.3. Notice that

in the model, for the sake of simplicity, the subscript of each transition and its label are the same except those highlighted, which represent transitions with the same label. The initial alignment α_b, its fitness π_{α_b}, and $\sigma_{N,\perp}$ is depicted in Fig. 5.14 (a), (b). For the moment we ask the reader to ignore highlighted parts.

The initial α_b is far from the optimal alignment. As mentioned already, this problem might be somewhat alleviated by reordering α_b, i.e., by merging the appropriate moves in log and model, to improve the corresponding fitness and preserve the executable property of the modeled trace simultaneously.

For example in Fig. 5.14 (a), the asynchronous moves $(a_3, \perp)$ and $(\perp, t_3)$, highlighted by green color, can be merged to make a synchronous move (a_3, t_3), highlighted with the same color in Fig. 5.14 (b). Notice that the resulted modeled trace remains executable. The same holds for asynchronous moves $(a_6, \perp), (a_8, \perp)$, which can be transformed to synchronous moves $(a_6, t_6), (a_8, t_8)$ by the corresponding merging which is highlighted accordingly. It is worth to note that $(a_{10}, \perp)$ and $(\perp, t_{10})$, highlighted with red, are not allowed to be merged to (a_{10}, t_{10}), because in that case the resulting modeled trace is not executable. Also note that $(a_{19}, \perp)$ and $(\perp, t_{18})$ can be merged without violating any conditions mentioned in Def. 30, as depicted in Fig. 5.14 (c) highlighted with gray. The same holds for $(a_{23}, \perp)$, which can be merged with its counterpart move in mode, i.e., $(\perp, t_{22})$. The corresponding merging is depicted in Fig. 5.14 (e). The last two merging reveals that transitions with the same label do not pose any challenges during the reordering of α_b. The final alignment and corresponding modeled trace, where no more reordering can be applied, are represented in Fig. 5.15 (a), (b).

To measure how much the fitness value is improved for the mentioned example, one can see that for the initial alignment α_b in Fig. 5.14 (a), based on Def. 18 (see Chapter 2, Sect. 2.2.3) with $\delta_S = 1, \delta_L = 2$, and $\delta_M = 2$, $\pi_{\alpha_b} = 1 - \frac{16\times2+18\times2}{11\times1+16\times2+18\times2} = 0.139$ and for the alignments depicted in Fig. 5.14 (c), (e) after merging highlighted moves are 0.188 and 0.25 respectively. Finally, the alignment α, see Fig. 5.15 (a), for which no more fitness improvement can be obtained has $\pi_\alpha = 1 - \frac{2\times2+4\times2}{25\times1+2\times2+4\times2} = 0.675$, which represents rather magnificent fitness improvement, even though it is not the optimal one. The optimal alignment α_{opt}, with $\pi_{\alpha_{opt}} = 0.764$, for the mentioned example provided by state of the art approach [5] is depicted in Fig. 5.15 (c). One can see that the only difference for this example between our approach and the approach in [5] is the synchronous move (t_{10}, t_{10}) in α_{opt}.

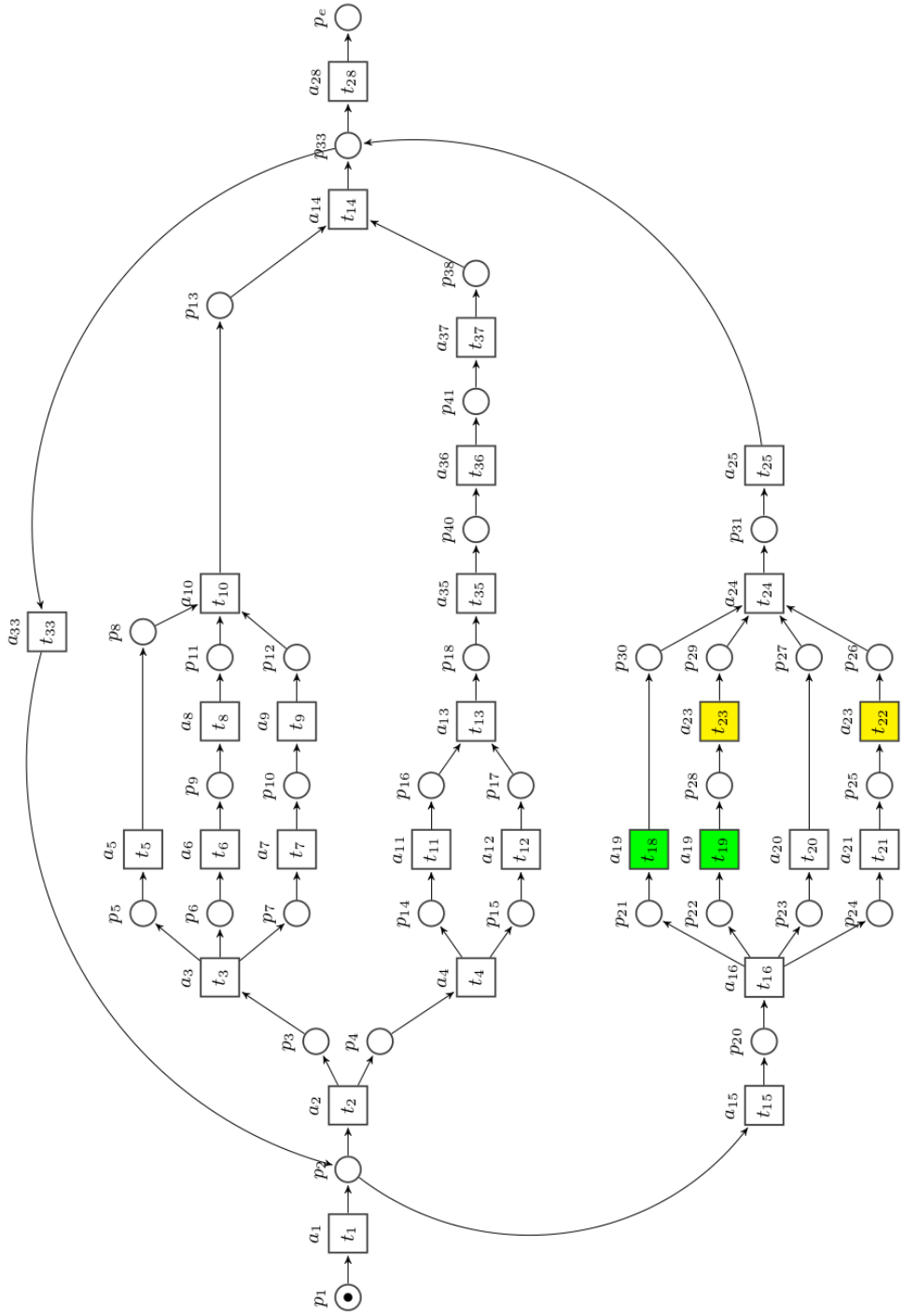


Figure 5.13: (a) Process model N_8, (b) Observed trace σ and modeled trace σ_N with and without labels

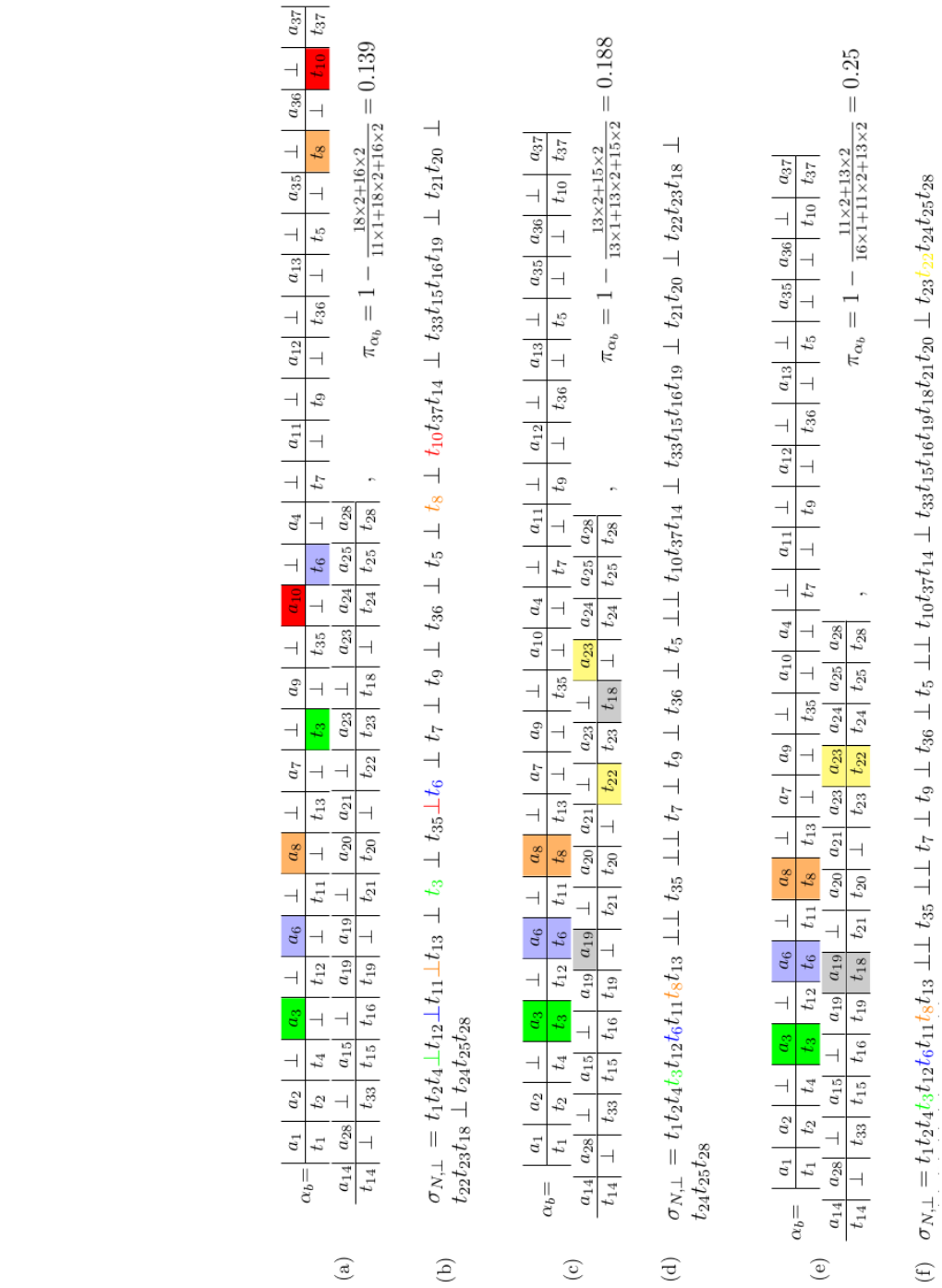


Figure 5.14: (a) Initial alignment, (b) Initial modeled trace including $\perp$, (c)-(f) Alignments and modeled trace after some adjustments

$\alpha=$

a_1	a_2	a_3	a_6	a_8	a_7	a_9	a_{10}	a_4	a_{11}	a_{12}	a_{13}	$\bot$	a_{35}	a_{36}	$\bot$	a_{37}
t_1	t_2	t_3	t_6	t_8	t_7	t_9	$\bot$	t_4	t_{11}	t_{12}	t_{13}	t_5	t_{35}	t_{36}	t_{10}	t_{37}

a_{14}	a_{28}	$\bot$	a_{15}	$\bot$	a_{19}	a_{19}	a_{20}	a_{21}	a_{23}	a_{23}	a_{24}	a_{25}	a_{28}
t_{14}	$\bot$	t_{33}	t_{15}	t_{16}	t_{19}	t_{18}	t_{20}	t_{21}	t_{23}	t_{22}	t_{24}	t_{25}	t_{28}

$$, \quad \pi_\alpha = 1 - \frac{2\times2+4\times2}{25\times1+2\times2+4\times2} = 0.675$$

(a)

$$\sigma_{N,\bot} = t_1 t_2 t_3 t_6 t_8 t_7 t_9 \bot t_4 t_{11} t_{12} t_{13} t_5 t_{35} t_{36} t_{10} t_{37} t_{14} \bot t_{33} t_{15} t_{16} t_{19} t_{18} t_{20} t_{21} t_{23} t_{22} t_{24} t_{25} t_{28}$$

(b)

$\alpha_{opt}=$

a_1	a_2	a_3	a_6	a_8	a_7	a_9	$\bot$	a_{10}	a_4	a_{11}	a_{12}	a_{13}	a_{35}	a_{36}	a_{37}	a_{14}
t_1	t_2	t_3	t_6	t_8	t_7	t_9	t_5	t_{10}	t_4	t_{11}	t_{12}	t_{13}	t_{35}	t_{36}	t_{37}	t_{14}

$\bot$	a_{28}	a_{15}	$\bot$	a_{19}	a_{19}	a_{20}	a_{21}	a_{23}	a_{23}	a_{24}	a_{25}	a_{28}
t_{33}	$\bot$	t_{15}	t_{16}	t_{19}	t_{18}	t_{20}	t_{21}	t_{23}	t_{22}	t_{24}	t_{25}	t_{28}

$$, \quad \pi_{\alpha_{opt}} = 1 - \frac{1\times2+3\times2}{26\times1+1\times2+3\times2} = 0.764$$

(c)

Figure 5.15: (a) Final alignment, (b) Final modeled trace including $\bot$, (c) Optimal alignment provided by state of the art approach [5]

Algorithm 10 Alignment Reordering

1: **Input: Global Variables** $SN(N, m_{start}, m_{end})$**,** α_b
2: prev-fitness $\leftarrow 0$
3: curr-fitness$\leftarrow \pi_{\alpha_b}$
4: **while** (curr-fitness $>$ prev-fitness) **do** ▷ Iterate until no fitness improvement can be made
5: **for** $(a_i, \perp)$ in α_b **do** ▷ Iterate over moves in log
6: **for** $(\perp, t_j)$ in α_b **do** ▷ Iterate over moves in model
7: **if** $(a_i = \ell(t_j))$ **then** ▷ Examine whether move on log and model match
8: **if** (**Merge**$((a_i, \perp),(\perp, t_j)))$ **then** ▷ Merging corresponding asynchronous moves
9: **Break** ▷ Break the current loop and start for another move on log
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: prev-fitness $\leftarrow$ curr-fitness
15: curr-fitness $\leftarrow \pi_{\alpha_b}$
16: **end while**
17: **Return** α_b

Algorithm 11 Merge

1: **Input:** $(a_i, \perp)$**,**$(\perp, t_j)$ ▷ Inputs are move in log and model respectively
2: $\sigma_{N,\perp}[i] \leftarrow t_j$ ▷ Flipping the corresponding elements to make a synchronous move
3: $\sigma_{N,\perp}[j] \leftarrow \perp$
4: **if** $(m_{start}[\sigma_{N,\perp}\rangle m_{end})$ **then** ▷ Examine whether the resulted modeled trace is executable
5: Update π_{α_b} ▷ Updating the fitness value based on the reordered α_b
6: **Return** 1
7: **else** ▷ Undo the flipping if $\sigma_{N,\perp}$ is not executable
8: $\sigma_{N,\perp}[i] \leftarrow \perp$
9: $\sigma_{N,\perp}[j] \leftarrow t_j$
10: **Return** 0
11: **end if**

The corresponding procedures required for the reordering of an alignment α are depicted by Alg. 10 and 11. The input alignment will be considered as the initial alignment, α_b. Given α_b, for each move in log, $(a_i, \perp)$, Alg. 10 finds the corresponding move in model $(\perp, t_j)$ with $a_i = \ell(t_j)$ to be merged, then examines whether the synchronous move (a_i, t_j) can be made with the use of flipping corresponding elements, lines 5-8. Whenever the candidate synchronous move is created without violating the conditions in Def. 30, then it breaks the current loop, line 9, and starts the same pro-

cedure for the another move in log[9]. Merging corresponding asynchronous moves can be accomplished successfully if the resulted modeled trace σ_N is executable, Alg 11, line 4.

5.6 Experiments

The proposed approach in this chapter has been implemented in Python 2.7 as a prototype tool[10] and Gurobi [32] was used as the LP solver. The standalone files for both Linux and Microsoft Windows operating systems can be downloaded from [73]. The tool has been evaluated over different family of examples from artificial to realistic containing transitions with duplicate labels and from well-structured to completely Spaghetti (see full descriptions of these datasets in Appendix A). The results are compared with the state of the art techniques for computing alignments [5](A^*), and a recent technique [67], from different perspectives which follow.

- **Execution Times Comparison**. Fig. 5.16 (a) and Fig. 5.16 (b), (c), (d) provide the required execution times for the mentioned datasets by the proposed approach, i.e., ILPSDP, and the approaches [5] (A^*) and [67] (DAFSA).

  One can see that for all benchmark datasets except M_8, M_9, ML_1 and *Banktransfer*, the proposed approach was faster than the A^* approach. For the DAFSA approach, when the state space becomes large (e.g., *Documentflow* or most of the rest of the examples) it cannot handle them. Hence, competency of the proposed approach becomes more clear in dealing with big models and long traces where the other approaches need to explore an exponential search space to the size of the given problem. As an example for *prDm6* the A^* approach runs out of memory (N/A in the figure) and the approach in this chapter can accomplish the task in reasonable time. By the same token, for realistic datasets the proposed approach is strongly faster on *Spaghetti* or *unstructured* models even when there are many loops in the model, see the results for *Documentflow1, Documentflow2.*

  Also, it is not difficult to grasp that the required execution time for the proposed approach is not as sensitive to the number of transitions with the same labels or size of the problem as the rest of approaches. Fig.5.16 (b) vividly represents this fact, see ML_3, ML_4 where the former model contains two transitions with the same label and later have six transitions with the same label.

[9] It is worth to mention that the approach considered in Alg. 6 is the simplest one to clear-up the idea in an algorithmic way, obviously for extremely large observed traces incorporating a hash function to map corresponding moves would be more efficient.

[10] The experiments have been done on a desktop computer with Intel Core i7-2.20GHz, and 8GB of RAM.

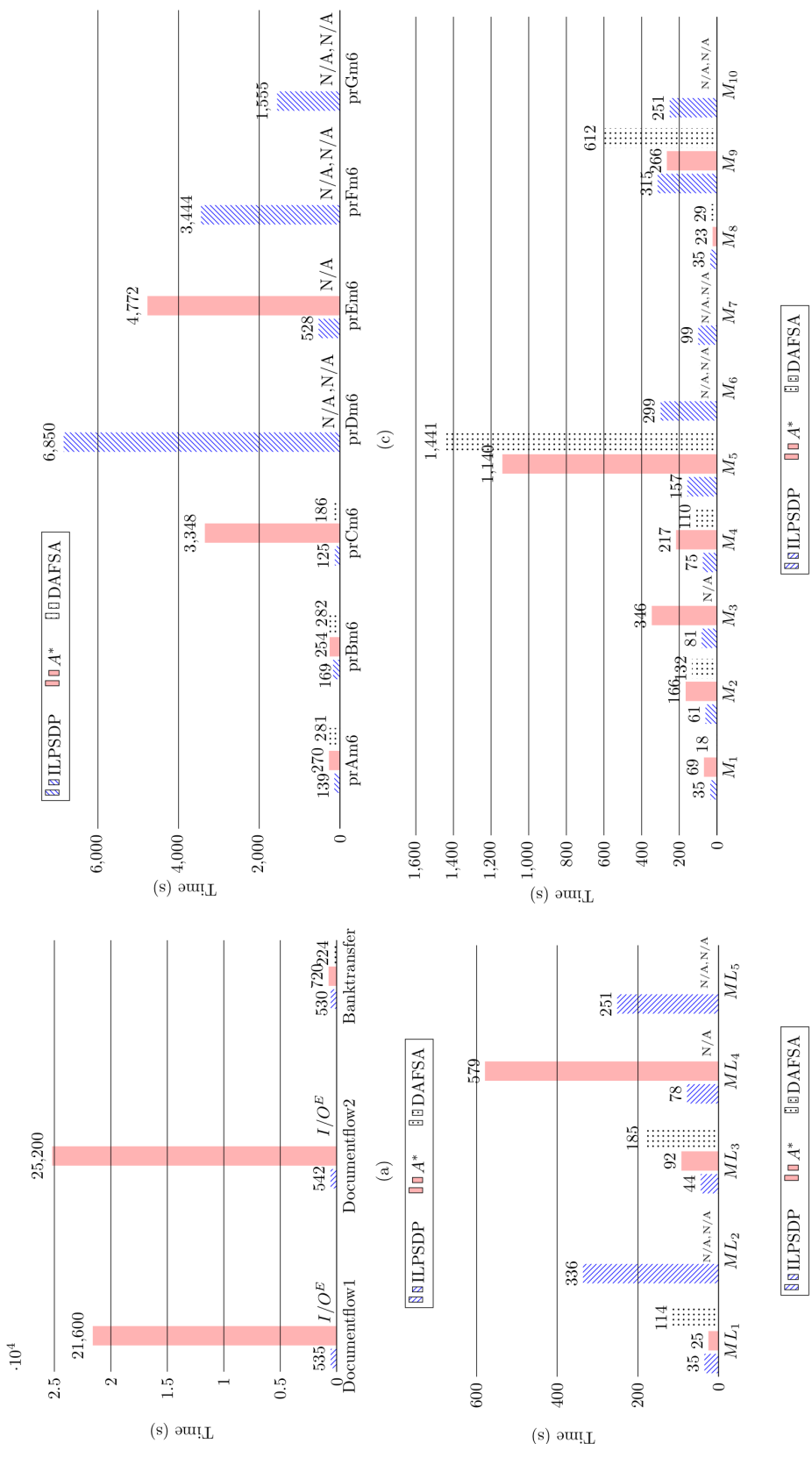


Figure 5.16: (a) Realistic dataset, (b) Synthetic datasets with duplicate labels, (c) Synthetic dataset [56], (d) Synthetic dataset, see Tables A.1, A.2, A.3, A.4, in Appendix A

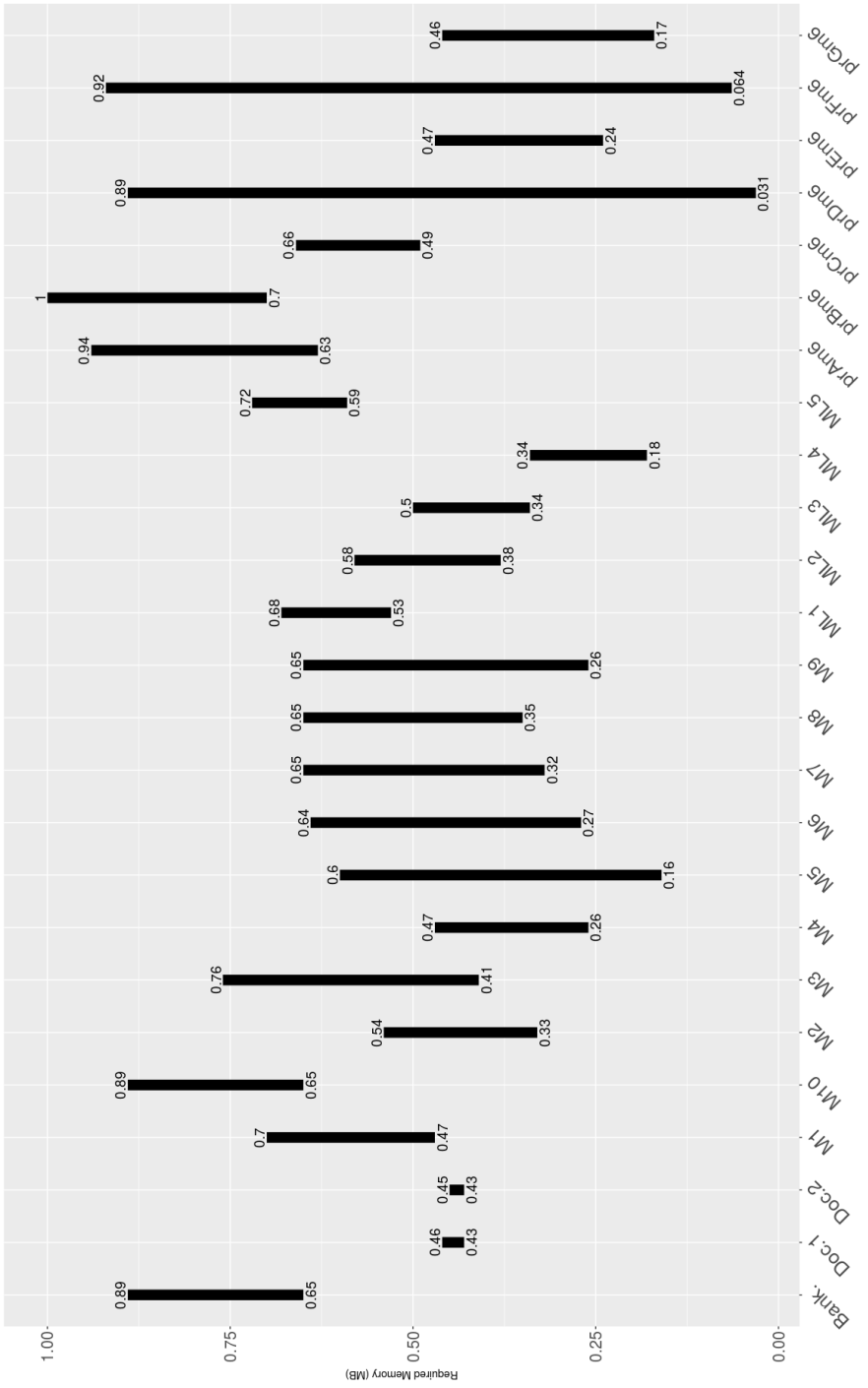


Figure 5.17: Fitness values improvement

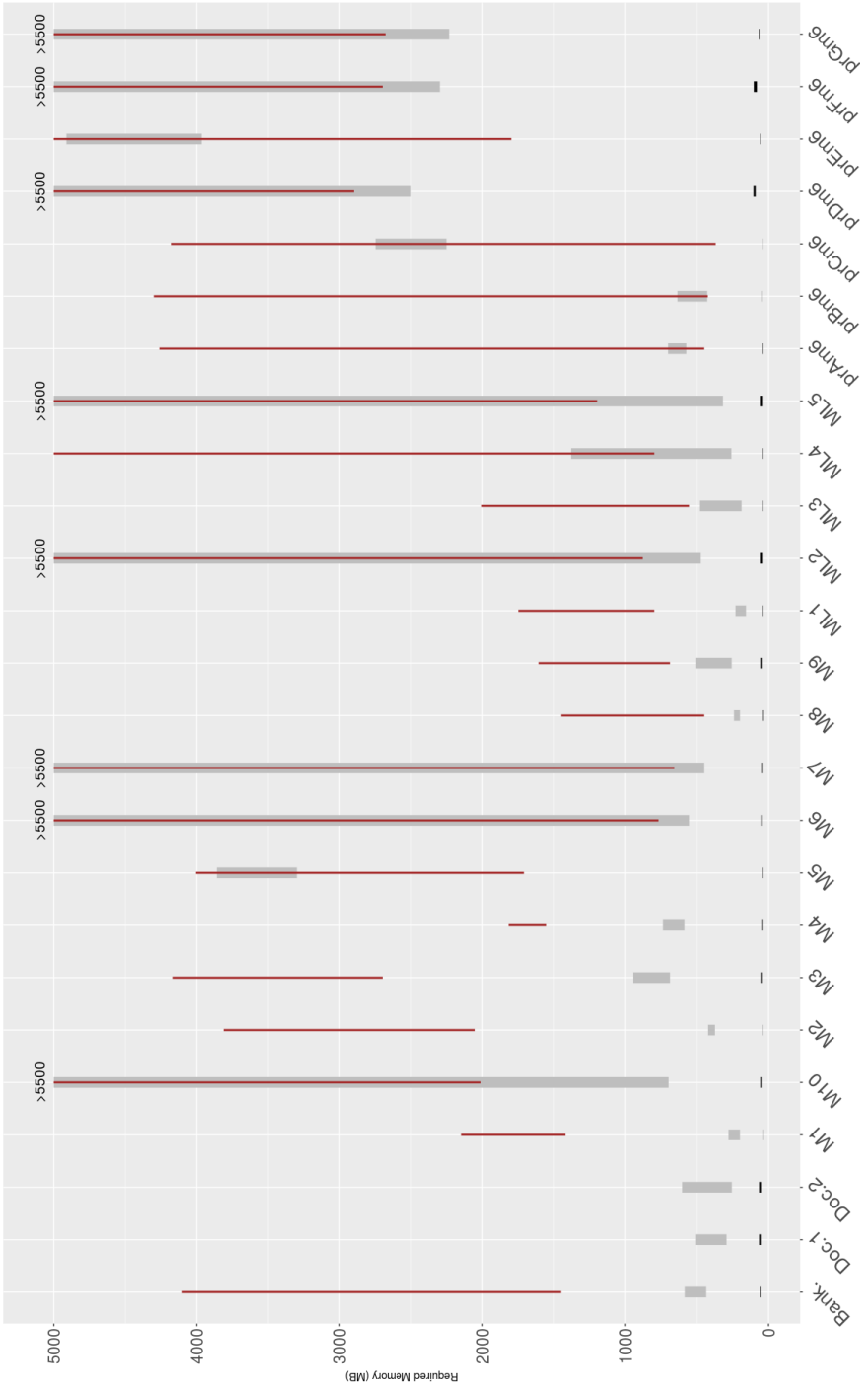


Figure 5.18: Memory usage of the proposed approach (black), the approach in [5] (gray) and in [67] (red)

Table 5.1: Comparison of fitness values

Model	Fitness MSE	Model	Ftiness MSE
prAm6	0.059	M_8	0.047
prBm6	0	M_9	0.022
prCm6	0.0038	ML_1	0.0152
prEm6	0	ML_3	0.034
M_1	0.0149	ML_4	0.051
M_2	0.029	Banktransfer	0.0007
M_3	0.034	Documentflow1	0.071
M_4	0.057	Documentflow2	0.069
M_5	0.037		

- **Fitness Comparison**. Table 5.1 represents the *mean square error* (MSE) of fitness values based on Def. 18, with $\delta_S = 1, \delta_L = 2$ and $\delta_M = 2$, between the computed alignments provided by the proposed technique and state of the art approach in [5] as optimal solutions, respectively[11]. These numbers simply quantify in average how close the respective fitnesses are from each other. The results were rounded to 4 digits. Comparisons were done only for those benchmark datasets whenever the approach in [5] could provide solutions. Overall, one can see that the approach of this chapter is very close to the optimal solutions computed by [5] in spite of, size of the model and observed traces and more specific in presence of loops, silent transitions and duplicate labels in the model.

- **Improvement of Fitness Values**. Fig. 5.17 reports the average of initial and final fitness values for the computed alignment before and after reordering based on Def. 30 for alignment reordering, the range of a fitness value represents the initial value for the base alignment α_b up to a stable alignment where no fitness improvement can be obtained. One can see that for large models and observed traces there is a magnificent jump for the corresponding fitness values and clearly models with massive parallelizations can get maximum benefits of the proposed technique, see *prDm6, prEm6* and *prFm6*.

- **Memory Consumption**. The memory usage of the proposed technique and state of the art approach in [5] and recent work in [67] for benchmark datasets are represented in Fig. 5.18. One can see that the proposed technique requires considerable less memory than state of the mentioned approaches for computing an alignment. Obviously for small and medium models, like *prAm6, prBm6, prCm6, Banktransfer*, the required memory for the proposed approach is at

[11] Since both [5] and [67] provide optimal solutions, we only need to compare the quality with any of these two approaches.

least 10 times less than the other approach and this ratio increases for large models with long traces. As an example, for *prDm6* the proposed method required around 105MB whereas the state of the art approach needs more than 5500MB[12]. Notice that the memory consumption of the proposed approach for computing an alignment is not sensitive to size of the model and length of the observed trace whereas state of the art approach in [5] is significantly sensitive due to expanding and exploring the corresponding search spaces, see prDm6, prFm6, prGm6, M_6, M_7 and M_{10}. Also, the required memory for the proposed approach is not sensitive to the labels of transitions i.e., silent or duplicate labels, see $ML_1, \ldots, ML_5$. One can see that for two benchmarks ML_3 and ML_4 with similar size in both model and event log but having different number of duplicate transitions, state of the art approach needs more memory for the one with more duplicate labels, i.e., ML_4 due to backtracking.

5.7 Outlook

A novel light and fast technique based on reordering of the initial alignment, which comes from replaying the resolution of ILP instance was proposed in this chapter. The proposed approach after obtaining an initial solution does iterate over it to find a better one. It must be mentioned that other heuristics can be applied to have an initial solution which definitely affects the next step. The technique has been implemented into a publicly available tool, and the evaluation shows promising capabilities to handle large instances including loops, silent transitions and transitions with duplicate labels. The evaluation reveals that this approach has good performance in different perspectives, and can deal with large and as well as Spaghetti and well-structured models.

[12] For each of benchmark datasets prDm6, prFm6, prGm6, M_6, M_7, M_{10}, ML_2 and ML_5 the required memory for [5] is more than 5500MB but due to limited amount of memory of the machine by which the experiments were done, the total required could not be measured.

Chapter 6

Genetic Algorithm Optimization Approach

6.1 Introduction

This chapter proposes an evolutionary technique to approximate multiple optimal alignments. We trade-off computation time for memory, i.e., assume that in some contexts, it is acceptable to spend more time in the computation, provided that the memory footprint is guaranteed to not exceed a given bound. To accomplish this, we encode the computation of alignments as a *genetic algorithm* (GA), where tailored *crossover* and *mutation* operators are applied to an initial population of candidate model explanations. This way, the derivation of a set of alignments is the result of genetic evolution.

The technique proposed has some weakness that should be reported: first, it can only provide optimal alignments when certain conditions are satisfied (variability in the population and genetic convergence). In practice, however, the number of iterations may be decided a priori, which may be insufficient for genetic convergence, and the initial population may not contribute to reach optimal solutions. Second, the number of optimal alignments obtained is in practice inferior to the real number of all optimal alignments, due to the dependence to the initial population and genetic convergence. Hence, the proposed technique only approximates several optimal alignments.

In spite of the approximation nature of the technique proposed, there is a clear value for several reasons: first, to obtain more than one model explanation of an observed trace may open the door to apply a posteriori root-cause analysis to identify the most likely explanation, as has been described in [41]. Second, the technique proposed represents the first algorithmic alternative to search for multiple optimal alignments, which can be applied on large instances under bounded memory. In the same way as GA provided an interesting perspective for process discovery [87, 16, 98], this work contributes to open a research direction for computing alignments

on the large. Third, in contrast to the A^*-based alignment technique, our technique is non-deterministic in providing alignments, so that two runs of the method may obtain different result. This may be very useful in *multi-perspective alignments* [46]: since control-flow is aligned before other perspectives, randomness in the generation of the control-flow alignment will enable the exploration of a broader solution space in the rest of perspectives.

The chapter is organized as follows: in Sec. 6.2 we describe the encoding as a GA of the problem of searching several best model explanations. The general framework for approximating multiple optimal alignments is described in 6.3. Tool support and experiments with various benchmarks are reported in Sec. 6.4. Finally, conclusions are reported in Sec. 6.5.

6.2 GA for Computing Several Explanations of Observed Behavior

GA starts by creating an initial population, and then combining the best solutions through operators, to create a new generation of solutions which should be better than the previous generation. As it will be noticed bellow, in some cases the evaluation of solutions will be adapted depending on the operator applied, so that the search for solutions can be better guided. A GA approach to a problem usually starts by encoding a solution which is called a *chromosome*, and define functions to evaluate how good it is. Next, generating the initial population of chromosomes and defining corresponding operators, i.e., *crossover* and *mutation*. In our setting, chromosomes will be potential model traces, which are combined through tailored crossover and mutation operators. Alg. 12 demonstrates in short the corresponding required steps adopted in this chapter for generating several explanations of an observed trace.

Algorithm 12 GA for Computing Model Explanations of an Observed Trace

```
1: Input: N, σ                          ▷ Inputs are model and log trace respectively
2: pop ← GENERATE(N, σ)                 ▷ Generating initial population
3: pop ← EVALUATE(pop, N, σ)            ▷ Rank chromosomes
4: while Not satisfactory do
5:     pop ← CROSSOVER(pop)             ▷ Applying crossover operators
6:     pop ← MUTATE(pop)                ▷ Applying mutation operators
7: end while
8: Return pop
```

Given an observed trace σ and WF-net N, a random population of chromosomes is first generated (Sect. 6.2.1). Then, it evaluates each chro-

mosome based on a specific fitness function[1], which considers both the initial model (for measuring replayability), and the observed trace (for measuring similarity) (see Sect. 6.2.2). It then applies traditional crossover and mutation operators, as well as novel ones defined for this problem, to speed up the process of evolving chromosomes and convergence (see Sect. 6.2.3). This process continues until reaching satisfactory results, or will be stopped by a predefined number of iterations. The detailed descriptions will be presented in the next sections.

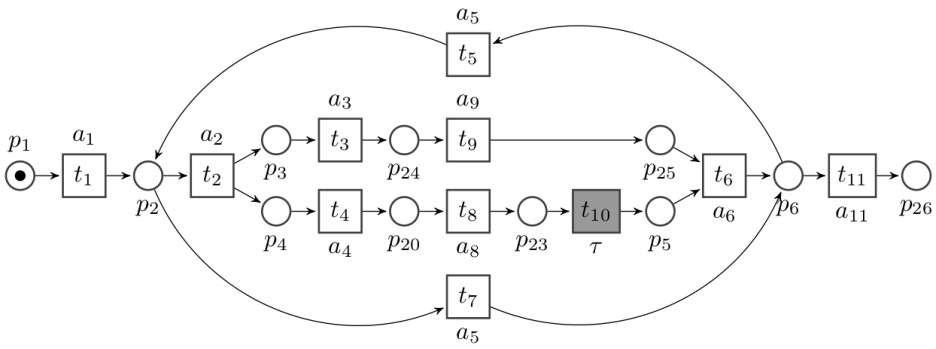


Figure 6.1: Process model M_2.

Before getting into the details, it is worth giving a big picture of one of the advantages that this approach makes. Fig. 6.2 shows the search space of a set of alignments with corresponding fitness values. Alignments with close fitness values are located on the same contour such that center of the plot shows alignments with better fitness values. The main objective of all alignment computation algorithms so far is to find an optimal alignment in this search space by exploring it. Stated differently, finding a point in this search space which can be considerably large. A GA is able to find an area in this search space which contains points with good (not exactly optimal) fitness values. In other words, this approach has a broader perspective than other approaches in search space exploration. This can be seen as a superiority since not only it decreases the probability of getting stuck in local optima, but also the algorithm might come up with more than one solution. The later issue would be attractive for process model repair techniques since it provides different angles of happened deviations.

6.2.1 Generation of the Initial Population

Given an observed trace σ, and WF-net N, the objective of this part is to generate an initial population. The population size is an important decision, which often affects the final solution in terms of accuracy and

[1] As the reader will soon realize, we refer to the term fitness in the genetic algorithms context.

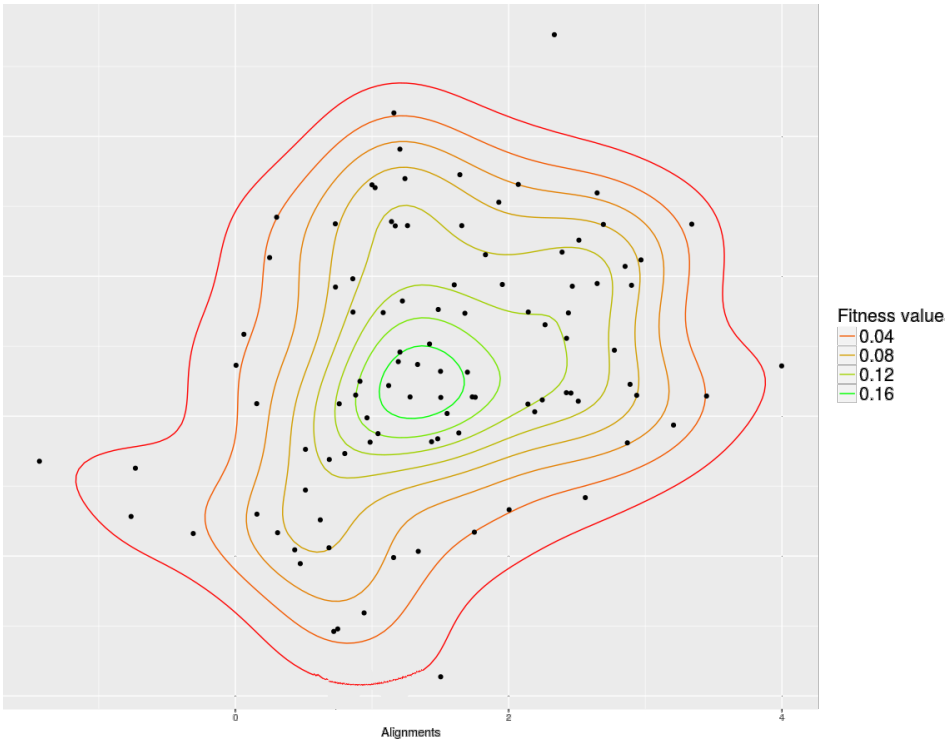


Figure 6.2: Alignments search space and respective fitness values

convergence [61]. Also, diversity in the population will help reaching different parts of the solution space. We rely on previous works for obtaining different model explanations, i.e., the methods presented in Chapters 5, 7 these methods are based on finding a maximal set of transitions, called *Parikh* vector that the model can reproduce to mimic the observed trace. A clear and detailed exposition is presented in Chapter 3, Sect. 3.4. More in detail, the *marking equation* of Petri nets is used to solve an (ILP) model for obtaining the corresponding Parikh vector. The ILP model has some additional constraints and a tailored cost function, that jointly guide the search for a maximal set with respect to its similarity (in terms of support) to the observed trace σ. To obtain the traces from the Parikh vector, we perform linearizations of the Parikh vector, obtained by either replaying it in the Petri net, or by arbitrary (possibly non-replayable) linearizations that do not consider the Petri net. Those are the seeds for generating new chromosomes. Formally a chromosome in this research is defined as follow:

Definition 31 (Chromosome)**.** *Let Parikh vector $\widehat{\sigma_P}$ be the solution of the optimization Eq. 3.2 in Chapter 3, then χ as a chromosome is the linearization of $\widehat{\sigma_P}$ such that $\widehat{\sigma_\chi} \leq \widehat{\sigma_P}$.*

For example consider the model in Fig. 6.1, and the observed trace $\sigma = a_1a_3a_8a_4a_2a_9a_5a_6$. Some chromosomes with respect to σ could be $\chi_1 = t_1t_7t_{11}$, $\chi_2 = t_1t_2t_3t_9t_4t_8t_{10}t_6t_{11}$, $\chi_3 = t_2t_1t_3t_4t_{10}t_6t_{11}$ and $\chi_4 = t_1t_7t_5t_{11}$.

$$\widehat{\sigma_P} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \widehat{\sigma_{\chi_1}} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \widehat{\sigma_{\chi_2}} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \widehat{\sigma_{\chi_3}} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \quad \widehat{\sigma_{\chi_4}} = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Figure 6.3: Parikh vector $\widehat{\sigma_P}$ and respective Parikh vectors of chromosomes

It is worth mentioning that some chromosomes may not be replayable at this stage (e.g., χ_4 above). The respective Parikh vectors are shown in Fig. 6.3.

6.2.2 Evaluation Criteria

In GA's jargon a fitness function is a particular type of objective function that prescribes the optimality of a solution (that is, a chromosome) in the corresponding population. Elevated chromosomes, which are the best ones at the corresponding time are allowed to breed and mix their datasets by any of several techniques, producing a new generation that will (hopefully) be even better. An ideal fitness function correlates closely with the algorithm's goal, and yet may be computed quickly. Speed of execution is very important, as a typical GA must be iterated many times in order to produce a usable result for a non-trivial problem.

In this chapter a chromosome χ is evaluated based on two metrics. The first one quantifies how good a chromosome is executable and the second one measures how much difference will be between the chromosome and the observed trace. Formal definitions of each operator are as follows.

Definition 32 (Executing fitness)**.** *For a given chromosome like χ, $f^m(\chi)$ denotes how good it is from execution perspectives. It is composed of three functions, the number of missed tokens while it is replayed, a binary function on whether the final marking is hit and number of tokens (remaining or missed) after execution. These functions are denoted by f^0, f^1 and f^2 respectively.*

$$f^m(\chi) = C_0.f^0(\chi) + C_1.f^1(\chi) + C_2.f^2(\chi) \tag{6.1}$$

Cs are constant that weight respective functions.

Each function in Def. 32 will be detailed as follows:

- $f^0(\chi)$: This function shows the ratio of *missed tokens*[2] to the total tokens while χ is being replayed in the model. In mathematical notation:

$$f^0(\chi) = \frac{\sum_{j=1}^{j=|\chi|}(m_j - \mathbf{N}.X_{j-1})^T.\overrightarrow{1}}{\sum_{j=1}^{j=|\chi|}(m_{j-1} + \mathbf{N}.X_{j-1})_+^T.\overrightarrow{1}}, \quad \forall t_j \in \chi, \tag{6.2}$$

  Function 6.2 fully takes advantage of the marking equation, i.e., Eq. 2.1 in Sect. 2.1.1.2. The numerator shows number of missed tokens when χ is replayed. Note that in each Parikh vector X_j only the element that is being replayed has value one and the rest are zero. Denominator computes the number of generated tokens when chromosome is replayed. The plus subscript means all elements of the obtained marking are considered as positive.

- $f^1(\chi)$: Although Eq. 6.2 gauges the executable property of a chromosome, though, there are some caveats that must be addressed. If during the execution of a chromosome it does not hit the final marking m_{end}, i.e., the following condition holds:

$$!\exists m_k \in \{m_j = m_{j-1} + \mathbf{N}.X_{j-1}\}_1^{|\chi|}, \quad \text{such that} \quad m_k^T.m_{end} = 1 \tag{6.3}$$

  Then that chromosome will receive another penalization. The argument behind this penalization is to give greater favor to chromosomes that hit the final marking.

- $f^2(\chi)$: Another issue that is overshadowed in evaluating a chromosome like χ, would be the number of missed and remaining tokens after it is replayed, i.e., when we $m_{|\chi|}$ marking is reached. It is important to consider these deficiencies as well, and distinguish them from those problematic tokens during the chromosome execution. Stated differently, these tokens show that the chromosome under consideration lacks of some genes to get the final marking. Thus a chromosome with this attribute receives extra penalty as follow:

$$f^2(\chi) = \frac{m_{|\chi|+}^T.\overrightarrow{1}}{dim(m_{|\chi|})} \tag{6.4}$$

  Where $dim(m_{|\chi|})$ shows the dimension of vector $m_{|\chi|}$ or simply the number of places in the respective model.

[2] In Petri net terms, missed tokens represent tokens that hamper the firing of a transition.

Definition 33 (Normalized edit distance fitness)**.** *For a given chromosome χ and an observed trace σ the normalized edit distance fitness between two mentioned sequences is defined as follow:*

$$f^{ed}(\chi) = \frac{Edit(\ell(\chi), \sigma)}{|\chi|} \tag{6.5}$$

Where $Edit(\ell(\chi), \sigma)$ shows the edit distance between two sequences $\ell(\chi)$ and σ, and $|\chi|$ denotes the length of the chromosome.

After defining the required metrics for evaluating a chromosome, totally they are composed to one single number as follow:

Definition 34 (Chromosome fitness)**.** *Given a chromosome like χ, let $f^m(\chi)$, f^{ed} be according to definitions 32 and 33 respectively, then fitness of χ is:*

$$f^t(\chi) = \lambda_1 \cdot f^m(\chi) + \lambda_2 \cdot f^{ed}(\chi) \tag{6.6}$$

Both λ_1 and λ_2 denote the penalization terms which will be adjusted individually for each genetic operator, and will be discussed in the next sections. It is clear that the lower value of $f^t(\chi)$ represents a better chromosome, i.e., modeled trace, by which the observed trace is mimicked. It should be pointed out that always having a chromosome χ with small $f^t(\chi)$ does not represent a desired or good solution if it is not replayable (i.e., $f^m(\chi) \neq 0$).

To get the idea of evaluation criteria, consider chromosome $\chi = t_1t_2t_9t_3t_8$ $t_4t_{10}t_{11}t_6$, the model in Fig. 6.1 and observed trace $\sigma = a_2a_1a_3a_9a_8a_4a_{11}a_6$; the number of missed and total tokens while χ is replayed equals to 3 and 23 respectively. Also, note that the final marking is marked therefore no penalization term applies from Eq. 6.2, i.e., $f^1(\chi) = 0$. Above that, the penalization that comes from Eq. 6.4 is 1, i.e., $f^2(\chi) = \frac{m^T_{|end|_+} \cdot \overrightarrow{1}}{dim(m_{|end|})} = \frac{1}{11}$.Thus $f^m(\chi) = \frac{3}{23} + 0 + \frac{1}{11} \approx 0.22$, see Fig. 6.4. Additionally, unreplayable transitions t_9, t_8 and t_{11}, are likely to be considered through genetic operators, in the next step of the proposed approach. The corresponding edit distance, i.e., $f^{ed}(\chi)$, between χ ans σ[3] is 5. Thereby, by selecting $\lambda_1, \lambda_2 = 1$, the corresponding fitness value is $f^t(\chi) = 1 * 0.22 + 1 * 5 \approx 5.22$.

[3] Note that indeed the edit distance is computed between σ and $\ell(\chi)$.

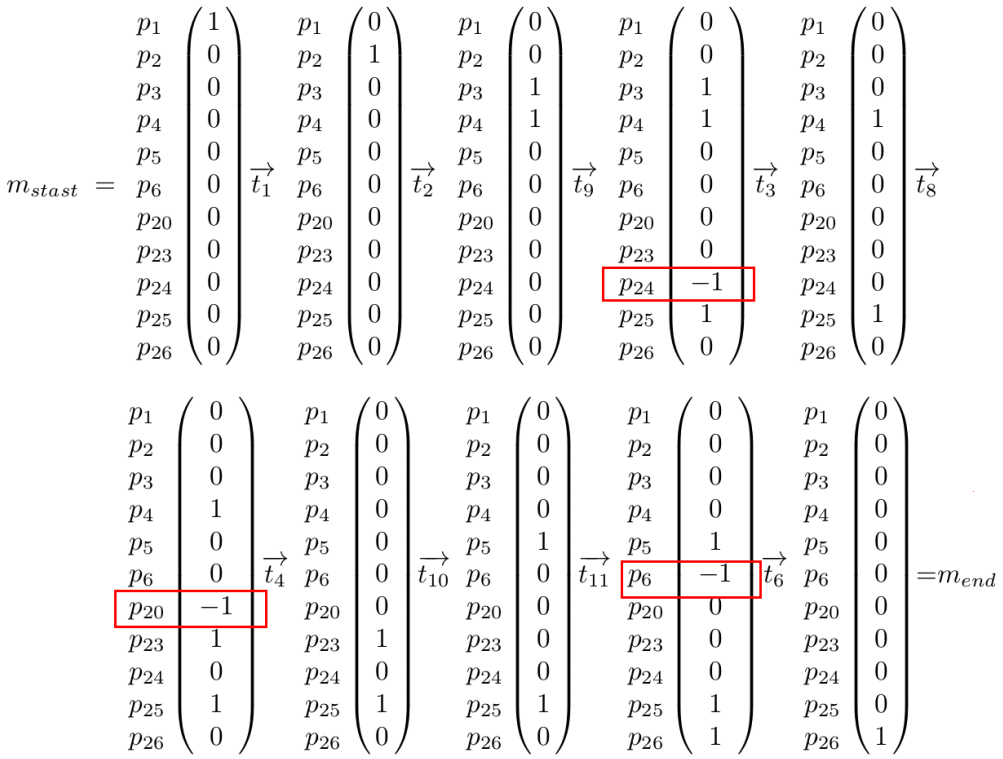


Figure 6.4: Replaying a chromosome

6.2.3 Genetic Operators

Genetic operators used in GA are analogous to those which occur in the natural world: survival of the fittest, or selection; reproduction (crossover); and mutation. When GA proceeds, both the search direction to optimal solution and the search speed should be considered as important factors, in order to keep a balance between exploration and exploitation in search space. In general, the exploitation of the accumulated information resulting from GA search is done by the selection mechanism, while the exploration to new regions of the search space is accounted for by genetic operators. In the remainder of this section, several genetic operators will be proposed. Some of them are inspired from ones found in analogous problems, whilst new ones are proposed that tend to improve the evaluation criteria described in the previous section.

6.2.3.1 Crossover operators

Crossover is the main genetic operator. It operates on two chromosomes at a time and generates two new chromosomes by combining both chromosomes features. A standard way to achieve crossover is to choose a random segment at both chromosomes, and generate two new chromosomes as the result of interchanging the two segments among the original

two chromosomes. In this chapter some context dependent operators have been proposed as follows:

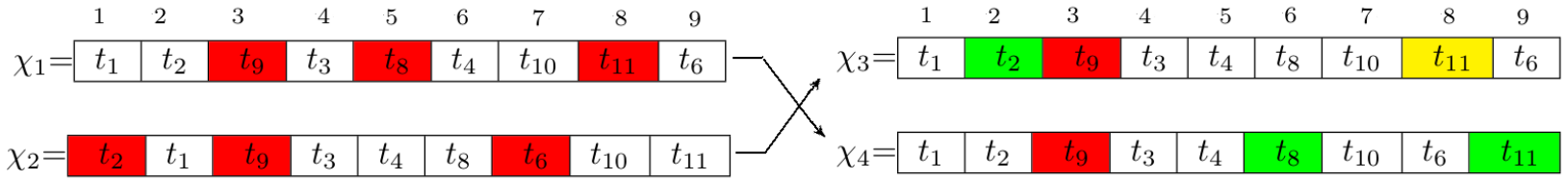
- **Modified Partially-Mapped Crossover (MPMX)**: We apply an adaptation of this standard crossover operator, for chromosomes having the same Parikh vector representation (see 2.1.1.2)[4]. The intuitive idea for operating over chromosomes with identical Parikh vector is due to the fact that the search space is reduced, and in particular the generation of the initial population is oriented towards satisfying this property. In order to keep Parikh vector representation of the original chromosomes, some modifications are done after the segments are interchanged. Let us look at the example in Fig. 6.5b to illustrate the MPMX operator; the initial chromosomes χ_1 and χ_2 are mixed with this operator, generating the new chromosomes χ_3 and χ_4, choosing a segment between positions 4 and 6.

  To keep the Parikh vector representation of the original chromosomes, some modifications are performed circularly starting from the first position after the segment (in the example, position 7). For instance, in χ_3 (that arised from χ_2 inserting the segment from χ_1), in the third position t_5 is removed since $|\chi_1|_{t_5} = 2$ and when we reach this position we already have 2 occurrences of t_5.

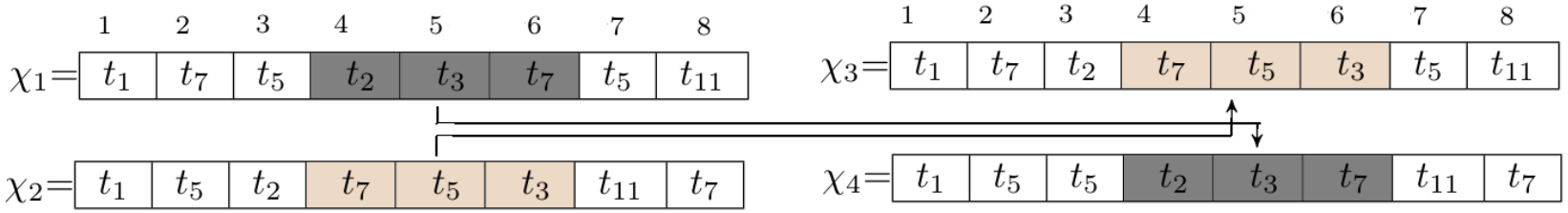
- **Cross-Insert Crossover (CIX)**: This operator tries to guide the search towards chromosomes that are replayable in the model. Still, the CIX operator works under the assumption of both initial chromosomes have the same Parikh vector representation. To induce replayability, it focuses on the parts of a chromosome that are not replayable, and uses the other chromosome in order to find candidate positions where it may be possible to reply the set of unreplayable transitions. This is done for each unreplayable transition in each one of the chromosomes that are merged. For each candidate position, the transition is moved to that position and the chromosome is shifted accordingly to fill the space left. For instance, let us look at the two chromosomes χ_1 and χ_2 in Fig.6.5a, and model M_2.

  In χ_1 the transitions in the third, fifth and the eighth position cannot be replayed, namely t_9, t_8 and t_{11}. Transition t_9 cannot be moved, since in both chromosomes it is unreplayable (so there is no candidate position in this case). However, for transition t_8 in χ_1 (which is at position 5) there is a candidate position (position 6, extracted from χ_2) to move. Moving t_8 to position 6 and shifting once from position 6 will leave the space in position 6 to put t_8 in χ_4. A similar situation happens with t_{11} position from χ_1 to the new position in χ_4. Notice that, as denoted in χ_3 in yellow, shifting may introduce new unreplayable

[4] The restriction on having the same Parikh vector is for the sake of simplicity of application.



(a) The CIX operator: in the figure, red background means unreplayable positions of the trace, while green denotes positions in the new chromosomes where unreplayable transitions have been fixed. Yellow denotes transitions that, in spite of being initially replayable, due to other moves, they became unreplayable.



(b) The MPMX operator.

Figure 6.5: Crossover operators

Algorithm 13 Cross-Insert Operator

1: **Input:** χ_1, χ_2 ▷ Inputs are two chromosomes with $\widehat{\sigma_{\chi_1}} = \widehat{\sigma_{\chi_2}}$
2: χ_3, χ_4= **Replicate**(χ_1, χ_2) ▷ χ_3, χ_4 are replicates of χ_1, χ_2 respectively
3: **for** $t_k \in$ **NoReplay**(χ_1) **do** ▷ t_k is not executable in χ_1 but it is χ_2
4: **if** $t_k \notin$ **NoReplay**(χ_2) **then**
5: index_correct= **Index**(χ_2, t_k) ▷ Position of t_k in χ_2
6: index_incorrect= **Index**(χ_1, t_k) ▷ Position of t_k in χ_1
7: **Swap** (χ_3, t_k, index_incorrect, index_correct) ▷ Change to the new position
8: **end if**
9: **end for**
10: ....
11: Do another loop for χ_2 and χ_4.
12: **Return** χ_3, χ_4

transitions: see t_{11}. Alg. 13 shows the implementation of this operator. The index operator in line 5 returns a random number among correct positions for t_k if there are more than one. The complexity of this operator is linear to the number of non executable elements in both chromosomes, i.e., $O(\text{NoReplay}(\chi_1)+\text{NoReplay}(\chi_2))$.

We stress that, since this operator tries to generate more executable offspring regardless of the corresponding edit distance, in our experiments we assign small values to λ_2 of Eq. 6.6, to retain the new generated chromosomes in next generations even if the edit distance has been degraded at the expense of improving replayability.

6.2.3.2 Mutation operators

Mutation applies to a single chromosome, generating a new chromosome as a modification of the initial one. It is viewed as a background operator to maintain genetic diversity in the population. Mutation helps escaping from local minimas trap and maintains diversity in the population. This part presents both generic and specific mutation operators related to the problem considered in this chapter.

- **Scramble Mutation (SM)**: This ia generic operator which simply chooses a segment in the chromosome, and randomly shuffles it. For instance, in Fig. 6.6 we show how the operator works. It selects a segment of a given chromosome randomly and reorder the elements of that segment.

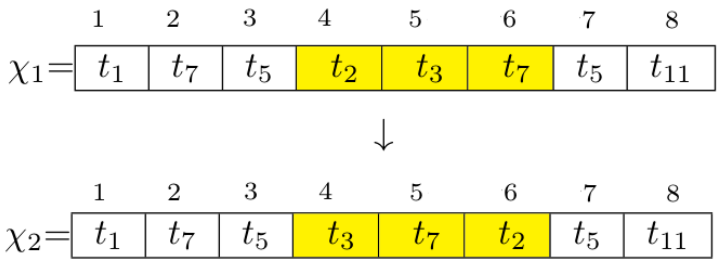


Figure 6.6: Scramble mutation operator (SM).

- **Mimic Mutation (MM)**: This is a specific operator proposed exclusively for the problem at hand. It tries to mimic the observed trace, by repositioning a transition t as close as possible to the position $\ell(t)$ that is observed in σ. Hence, this operator tends to reduce the edit distance between σ and χ by repositioning t. For example, assume that we want to relocate t_i in χ to the position of a_i in σ where $\ell(t_i) = a_i$. To decrease the edit distance between σ and χ, we can do mirroring the respective elements, i.e., a_i, t_i. Doing so, it might end up with a fatal error if the corresponding event in the observed trace is a deviation. To tackle this challenge, we resort to information available in other observed traces to estimate approximately where t_i can be relocated. To this end, the positions of corresponding element, i.e., a_i will be extracted from the other observed traces, and after that the corresponding histogram is created. This histogram shows the distribution of positions that a_i can have. It can be interpolated to obtain a respective curve. Since interpolating the histogram in our situation is very sensitive to noise, instead we adopt a *smoothing* mechanism to alleviate this effect, and obtain a more stable density function. Smoothing means decreasing the effect of seen events and increasing the effect of unseen events. In mathematical notation it is defined as follow:

  Definition 35 (Smoothing histogram)**.** *Given an observed trace σ, let assume that event a_i is in position x_i, then for a_i the corresponding smoothed frequency is:*

$$y_i = f_{a_i}(x_i) + \epsilon_i, \quad \forall x_i \in L \tag{6.7}$$

  Where x_i is defined over the positions that a_i takes in the whole set of observed traces or event log L. y_i shows the smoothed frequency of the respected position x_i. Also, the difference between actual and smoothed frequency is packed into ϵ_i. Function $f_{a_i}()$ is a smoothed curve that must be estimated or learned accordingly.

It must be stressed that, f_{a_i} varies for different element a_i. Implementing this operator has two main steps. First, estimating or learning the smoothed frequencies, and second generating a random number from the estimated curve. To proceed for obtaining smoothed frequencies, for the sake of simplicity assume that $x_1, x_2, ..., x_n$ are positions of element a_i in L, where to each position a frequency number is assigned. To estimate $f_{a_i}()$:

1. One can divide $x_1, x_2, ..., x_n$ into many equal-width bins. The plot of bin centers versus bin heights is a rough estimate of $f_{a_i}()$. Assume that the probability of having a point in bin $[x_i, x_{i+1}]$ is p_i then, the likelihood of the given histogram is proportional to multinomial likelihood $\prod_{j=1}^{j=B} p_j^{y_j}$[5], where B is the number of bins. Also, it is equivalent to work with B separate *Poisson* distributions with expectations $\mu_i = np_i$ [25], [69]. The previous conclusion directs us to use *Generalized Linear Model* (GLM) for smoothing. More precisely, assume that x_{min} and x_{max} are the minimum and maximum of $x_1, x_2, ..., x_n$ respectively, and suppose that we adopt B histogram bins on the interval $[x_{min}, x_{max}]$ so that bin widths are $\Delta = \frac{(x_{max} - x_{min})}{B}$. Let N_j be number of points and let c_j shows the center of bin. Since the area under a density function is 1, then the height of jth bin is $y_j = \frac{N_j}{n.\Delta}$. To make a smooth estimation we can regress N_j on the c_j using Poisson regression spline. The fitted curve, i.e., $\widehat{N_j}$ then will be divided by $n\Delta$ to yield the smooth density estimate, i.e., $\hat{y}_i$.
2. After estimating the corresponding density for a_i then we can generate a random number out of it by which t_j will be relocated to that position. There are many algorithms for generating random numbers from an arbitrary distribution, like *inverse transform sampling* [100].

To derive the concept home, a concrete example is provided next. Assume that χ_1 and σ_1 are given inputs to this operator as depicted in Fig. 6.7 (a), (b). The task is, Element t_9 in χ_1 will be mutated so as to mimic a_9. To this end, we use the available information in other 1000 observed traces to figure out, how this element, i.e., a_9 is distributed across other observed traces. The corresponding histogram is shown in Fig. 6.7 (d). The next step, is to find the center of bins which are plotted by black dots in Fig. 6.7 (e) (in this figure 200 bins were executed). The last step is to learn a smoothed curve according to the mentioned dot points, which can be done by using a Poisson regression between center of bins as regressors and height of bins or frequencies as the target variable. The estimated

[5] Note that y_i here is the number of points in the respective bin.

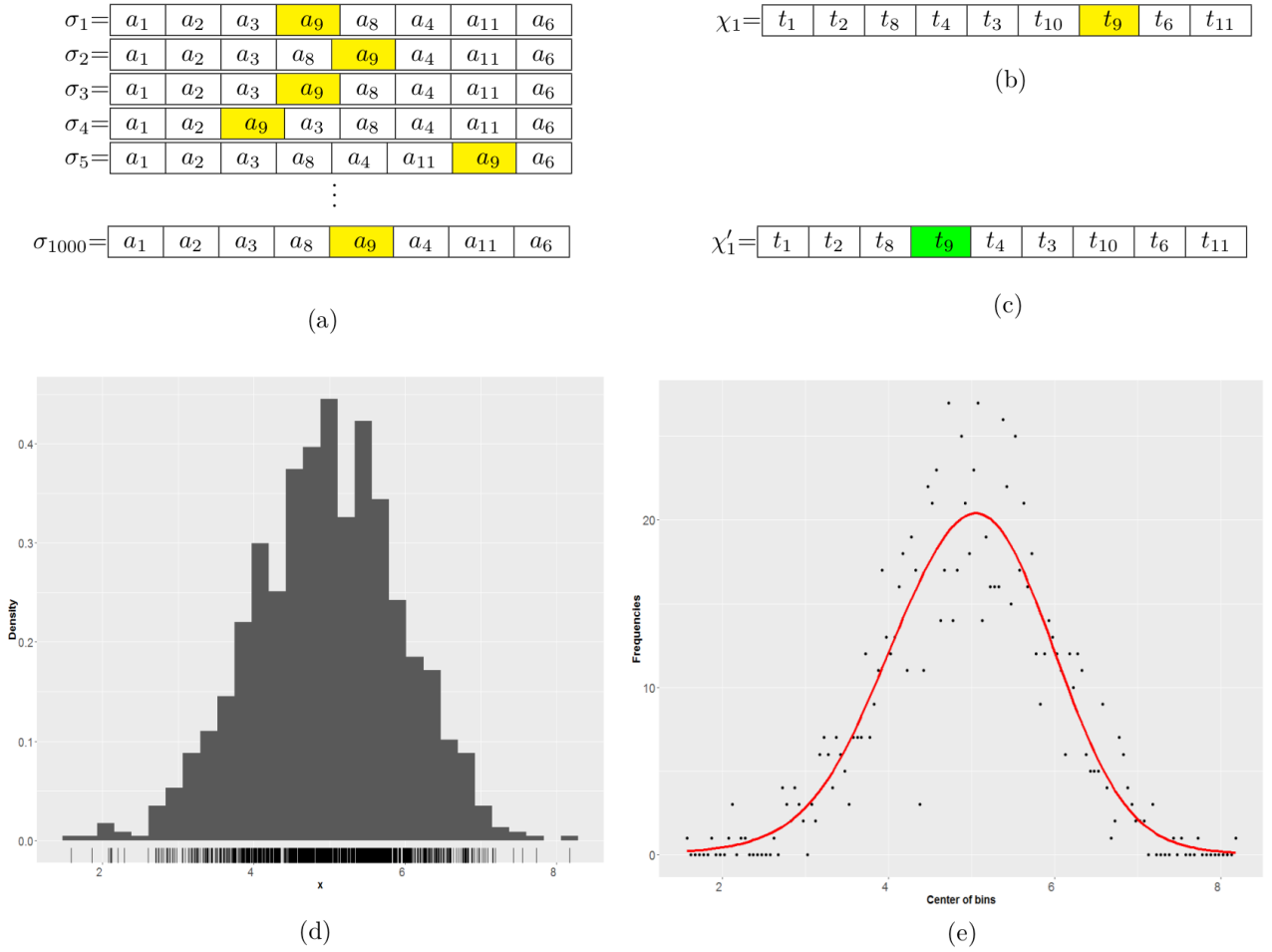


Figure 6.7: Mimic mutation operator (MM) (a) Event log, (b) Chromosome, (c) Muted chromosome, (d) Histogram of positions, (e) Smoothed frequency of positions

curve is shown in Fig. 6.7 (e) with the red line. This curve shows the behavior of element a_9 across observed traces. Whenever that curve is learned, a random number is generated from that and the respective element of χ_1, i.e., t_9 will be relocated to that position.

It is easy to see that, the more deviations observed in the event log the wider distribution we would have. It is important to note that, the learned curve is not necessary to be a bell-shaped, and it can be of any shapes. This is why we resorted to nonparametric regression techniques to consider this conditions. The other expected shapes that usually happen are depicted in Fig. 6.8 (a), (b). In spite of showing the behavior of an element, these plots can show other useful information, for example Fig. 6.8 (a) sheds light on having a change detection probably, since the distribution is very skewed, or Fig. 6.8 (b), suggests different scenarios, like having a loop in a model or existing duplicate transitions or in worst case it can be a concept drift for executing this activity which needs further investigations.

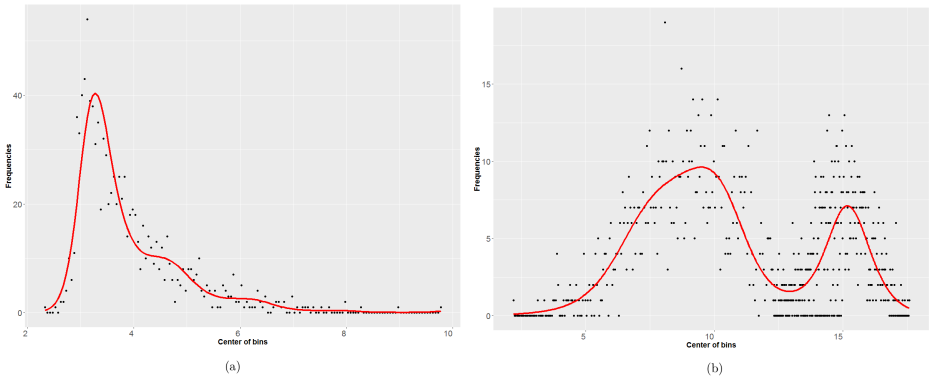


Figure 6.8: Learned curves of mimic mutation.

It is important to note that for each event element a_i a unique curve must be learned but this learning process needs to be done only once as a preprocessing step and whenever this operator is invoked, it takes advantage of them.

Finally, for a chromosome χ, and its offspring χ', since the goal is to mimic the observed trace, if $f^m(\chi) < f^m(\chi')$ and $f^{ed}(\chi) > f^{ed}(\chi')$ then λ_1 and λ_2 in Eq. 6.6 are adjusted so that sometimes χ' survives in the next iteration. In other words replayability is overshadowed for this operator. Playing with these parameters would decrease the risk of getting stuck in a local optimum. Alg. 14 shows the required steps to execute this operator. Note that in line 9, if the random generated number does not work for the chromosome under consideration, for example because of different length, then another random number will be generated. This operator is super light because it only needs to swap an element from an old to a new position.

Algorithm 14 Mimic Mutation Operator

Input: χ, σ ▷ Inputs are a chromosome and an observed trace
a_i= **Random**(σ) ▷ Selecting a random event
if $a_i \notin \ell(\chi)$ **then**
    No mutation is done
    Return ▷ If the corresponding element is not exist in χ mutation can't be done
else
    χ'= **Replicate**(χ) ▷ Replicating the chromosome
    index_current= **Index**(χ, t_k) ▷ Position of t_k in χ, given $\ell(t_k) = a_i$
    index_new = **Random** ($f_{a_i}()$) ▷ Generating a random position
    Swap (χ', t_k, index_current, index_new) ▷ Change to the new position
end if
Return χ, χ'

- **Launch Mutation (LM)**. Up to this point, none of the mutation operators above try to improve the replayable property of chromosomes. The intuitive idea is, for a given chromosome with an unreplayable position i, the transition in i will be relocated forward (i.e., in a position $j > i$) by this operator. The rationale behind this policy comes from the idea that in some situations, by delaying the firing of a transition to a future Petri net marking, enough tokens will be placed by the transitions occupying positions between i and $j - 1$. Since the overall goal of this operator is to improve replayability, we set $\lambda_1 < \lambda_2$ so that replayability has more importance to decide survival for the next iteration.

  To give a concrete example consider the chromosome in Fig. 6.9 (a), and the model M_2. Unreplayable transitions are highlighted (positions 3, 5 and 8). Assume that t_9 is selected to be mutated with is operator. Fig. 6.9 (b) shows one possible launch mutation from position 3 to position 7. One can see that transition t_9 can now be replayable, as highlighted in green. Unfortunately, this operator can sometimes introduce new unreplayable transitions, as demonstrated in Fig. 6.9 (c) for t_{10}.

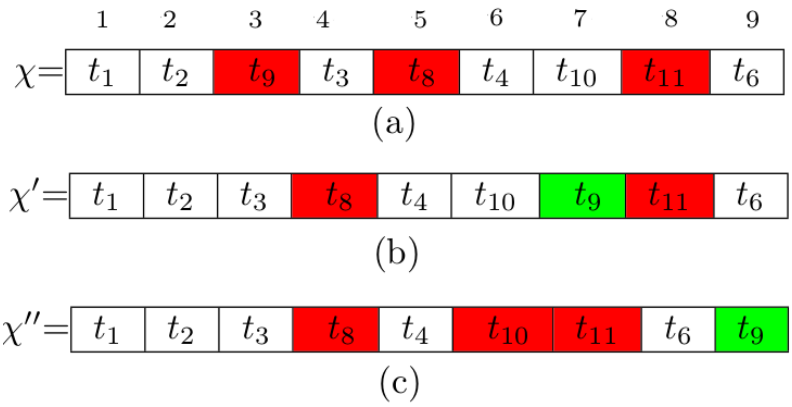


Figure 6.9: (a) χ, (b) χ' after LM on χ to position 7, (c) χ'' after LM on χ to position 9.

6.3 General Framework for Obtaining Multiple Alignments

Given a process model represented as a WF-net, N, and a trace σ, the schema of the proposed framework is depicted in Fig. 6.10. Explanations of each part are provided below:

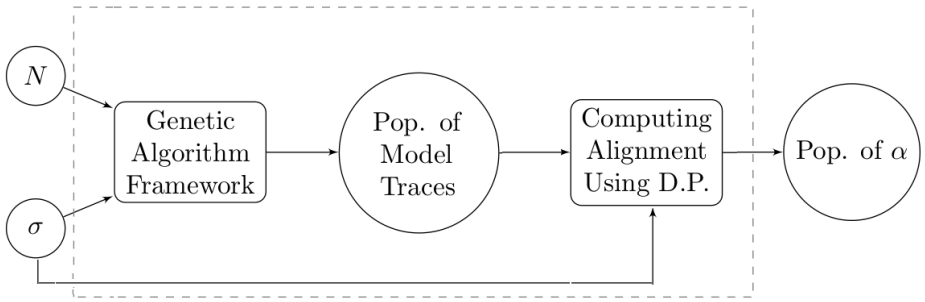


Figure 6.10: Overall description of the general approach to compute alignments.

- *Genetic Algorithm Framework*: In the initial stage, the genetic approach described in the previous section is performed. Once finished it generates a final population of model traces. Among them, we choose those chromosomes χ having both $f^m(\chi) = 0$ (so, replayable), and minimal $f^{ed}(\chi)$.

- *Computing Alignment Using Dynamic Programming*: This part concerns the computation of alignments between the chromosomes of

final population and σ. The adopted method in this section is a dynamic programming approach inspired from aligning two sequence of genes [58], [59] and the exposition is in Chapter 5, Sect. 5.4. The alignments computed are called *best alignments*, which are not necessarily optimal: this is due to the lack of guarantees that the model explanations provided in the previous stage correspond to the optimal model explanation for σ.

6.3.1 Computing an Alignment using Dynamic Programming

To compute an alignment between a chromosome like χ and observed trace σ, the technique presented in this chapter is inspired from [59]. This technique was already explained in details in Chapter 5, Sect. 5.4 for the same task, so we informally describe it here as a refresher. Consider an oversimplified example, $\chi = t_3 t_{11} t_{17}$ with $\ell(\chi) = a_3 a_{11} a_{17}$ and $\sigma = a_3 a_{11}$. To obtain an alignment α between these two sequences, a two-dimensional table is created, where the first row and first column are filled with the observed trace and chromosome, respectively, as depicted in Fig. 6.11 (a). The second row and second column are initialized with numbers starting from 0,-1,-2,..., they are depicted in yellow color. The task then is to fill the remaining cells with the recurrence Eq. 5.1 in Chapter 5 where for the sake of simplicity is repeated here. In this equation δ represents the *gap penalty*[6] and $s(t_i, a_j)$ represents both the match and mismatch cost between two elements t_i and a_j which are modeled and observed trace elements, respectively.

$$SIM(t_i, a_j) = \text{MAX} \begin{cases} SIM(t_{i-1}, a_{j-1}) + s(t_i, a_j) \\ SIM(t_{i-1}, a_j) - \delta \\ SIM(t_i, a_{j-1}) - \delta \end{cases} \qquad s(t_i, a_j) = \begin{cases} \beta & \text{If} \quad \ell(t_i) = a_j \\ -\beta & \text{If} \quad \ell(t_i) \neq a_j \end{cases} \tag{6.8}$$

$SIM(t_i, a_j)$ represents the similarity score between t_i and a_j.

[6] The gap penalty represents asynchronous move in our setting.

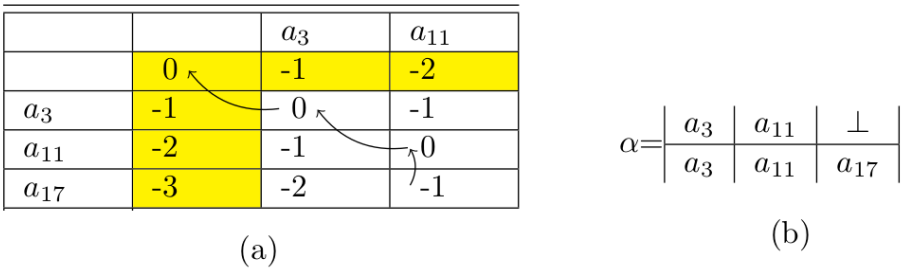


Figure 6.11: (a) Computing alignment using dynamic programming (b) Obtained alignment.

After filling the matrix, to compute the alignment we start from the bottom right entry, and compare the value with three possible sources, i.e., top, left and diagonal to identify from which one of them it came from. If it was fed by a diagonal entry, it represents a synchronous move between corresponding elements and if it was fed by a top or left entries then it represents an asynchronous move or a gap. For the mentioned chromosome and observed trace the computed alignment is shown in Fig. 6.11 (b).

6.4 Experiments

The approach of this chapter has been incorporated into the tool `ALI` [73]. This section evaluates the method proposed over the following perspectives:

- What is its sensitivity on the number of evolutionary iterations ?
- How does it compare to [5, 66] for the memory and execution time ?
- How does it compare to [5, 66] for the quality and quantity of alignments obtained ?
- What is the impact on the fitness calculation ?

The tool has been evaluated over different family of examples from artificial to realistic, containing transitions with duplicate labels and from well-structured to completely Spaghetti[7]. The number of transitions varies between models, i.e., minimum 15 and maximum 429. The full description of datasets are presented in Appendix A. We also included a real-life benchmark from [20], where a model was discovered using the Inductive Miner by sampling 10000 observed traces of a *Road Traffic* process dataset[8], and using the rest of the log for alignment computation. The results on these benchmarks are compared with the state-of-the-art technique for computing one optimal alignments [5], since the version for computing all optimal

[7] The experiments have been done on Intel Core i7-2.20GHz computer with 8GB of RAM.
[8] https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

alignments ran out of memory for all models considered in this chapter. We also compare `ALI` with a recent technique to compute all-optimal alignments [66].

- **Configuration of the Genetic Algorithm**. Since the decision of the initial population and the probabilities of operators strongly influences the genetic algorithm, we have chosen to customize it so that diversity is kept in early stages of the genetic evolution. It should be stressed that in the general application of genetic algorithms, populations are significantly larger than the ones considered for this chapter for the proposed approach, so that the probabilities for operators have been set accordingly, to avoid that few high-ranked chromosomes dominate the rest in early stages of the genetic evolution. In the implementation, the application of crossover had a high probability (80%, and then tuned with the individual probabilities set by parameters λ_1 and λ_2), whilst mutation operators were applied with a low probability (5%). In any case, the chosen probabilities are common when applying genetic algorithms in other scenarios. As we commented in the previous section, best alignments in our setting correspond to replayable chromosomes with minimal edit distance to the observed trace among the final population. For each observed trace a population of 700 chromosomes was generated, and the quality and quantity of them were compared with state of the art approach at iterations 10,20,30 and 100. We also experimented with population sizes significantly smaller (e.g., 100 chromosomes), and the results obtained were proportional in the main perspectives considered in this chapter: less memory footprint and execution time, but slightly worst quality.

- **Execution Times**. Fig. 6.12 shows violin plots of execution time (in seconds) for each model per iteration given an observed trace. Obviously the required execution time varies from different observed traces and this is why the corresponding distributions via violin plots are presented. One can see that for big models with large traces (*prDm6, prEm6, prFm6*), models with many deviations in observed traces (*prCm6*) and models with many duplicate transitions (ML_5), the corresponding distributions are wider due to more operations made by the proposed operators at any iteration. An important point should be done: although the computation time per trace (corresponding to multiplying the execution time per iteration shown in the plot by the number of iterations performed) is significantly higher with respect to [5], our evaluation is done with a simple, unoptimized implementation of the technique of this chapter.

- **Fitness Comparison**. Table 6.1 represents the *mean square error* (MSE) of fitness values (see Def. 18 in Chapter 2), among the best

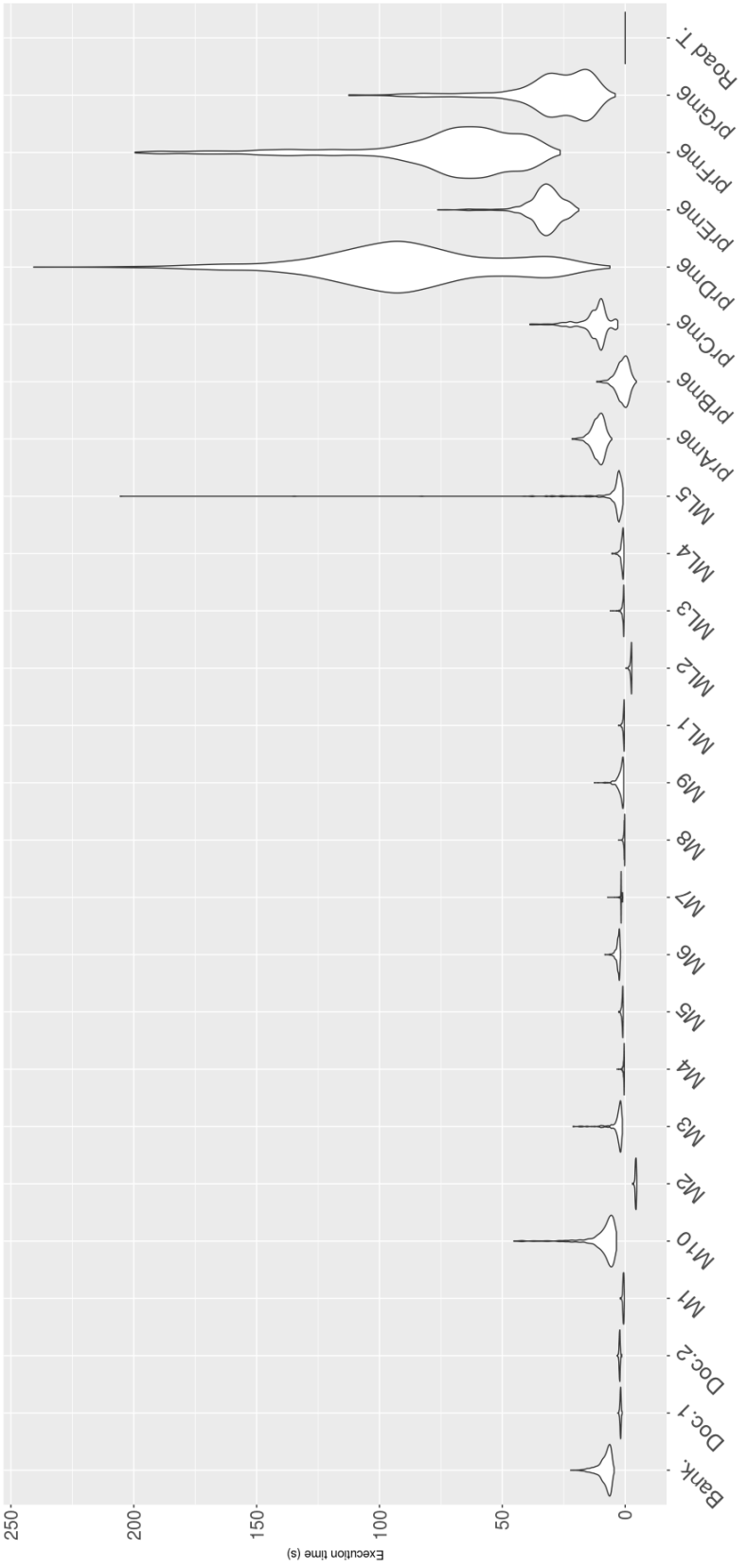


Figure 6.12: Distribution of execution time for an observed trace per iteration.

Table 6.1: MSE comparison of fitness values of chromosomes at different iterations and A^* as the optimal one

Model	It.(10)	It.(20)	It.(30)	It.(100)
prAm6	0.0106	0.0055	0.0035	0
prBm6	0.0009	0	0	0
prCm6	0.0776	0.0044	0.0031	0.0012
prEm6	0.0436	0.0353	0	0
M_1	0.0571	0.0089	0.0084	0.0046
M_2	0.0475	0.0116	0.0094	0.0070
M_3	0.0351	0.0341	0.0327	0.0219
M_4	0.1980	0.0512	0.0508	0.0398
M_5	0.0958	0.0289	0.0226	0.0155
M_8	0.0836	0.0384	0.0379	0.0357
M_9	0.0725	0.0220	0.0214	0.0203
ML_1	0.0775	0.0400	0.0146	0.0091
ML_3	0.1864	0.0991	0.0159	0.0142
ML_4	0.3113	0.1740	0.0330	0.0251
Bank.	0.0013	0.0008	0.0005	0.0002
Doc.1	0.2912	0.1971	0.0702	0.0698
Doc.2	0.2581	0.1701	0.0579	0.0570
Road T.	0.0036	0.0022	0.0018	0.0014

alignments provided by our technique and the approach in [5] as optimal solutions, respectively. One can see that the quality of alignments is improved from 10 iterations to 100 iterations for all models. For models $prAm6$, $prBm6$ and $prEm6$ optimal alignments were found (for some of them, at iterations 2 and 3 respectively). Comparisons were done only for those benchmark datasets whenever the approach in [5] could provide solutions. Overall, one can see that the approach of this chapter is very close to the optimal solutions computed by [5], in spite of several factors like the size of the model and observed traces, presence of loops, silent transitions and duplicate labels in the model.

- **Quantity of Best Alignments**. Table 6.2 shows the average number of best alignment obtained per each observed trace and each model at different number of iterations. In the last column, we report this number for [66]: NA denotes that the tool was unable to provide the result due to memory problems. When it can, we also provide in parenthesis the percentage of the log traces where [66] can find solutions; for instance, for M_4, only 39% of the traces have a solution. One sees that these average numbers are improved from

10 to 100 iterations and this improvements are usually more tangible in models containing loops, i.e., M_1, M_2, M_3, M_7. The approach from [66] usually obtains more alignments than our method, but that only holds for small or medium instances.

Also, Fig. 6.13 and 6.14 show for each model, the violin plots or distribution of number of best alignment for 30 and 100 iterations, respectively (the corresponding average values are shown in the fourth and fifth column of Table 3, respectively). When focusing in the experiment for 100 iterations (Fig. 6.14), It can be seen from the plot that, for some models like M_1, M_2 M_5, M_7 and ML_2, the number of distinct best solutions are close to 30 for some cases and for *Documentflow2* the best solutions are unique.

Table 6.2: Number of different alignments for the best solution found in average for the approach in this chapter and the approach in [66]

Model	It.(10)	It.(20)	It.(30)	It.(100)	ATM. $A*$[66]
prAm6	1.11	1.21	1.25	1.45	NA
prBm6	1.00	1.15	1.15	1.34	NA
prCm6	1.16	1.52	1.79	3.46	NA
prDm6	1.11	1.33	1.57	1.71	NA
prEm6	1.16	1.43	1.52	1.56	NA
prFm6	1.03	1.17	1.36	1.46	NA
prGm6	1.08	1.30	1.49	1.78	NA
M_1	1.94	2.91	3.32	4.32	62.12(92%)
M_2	2.98	4.97	5.89	7.13	320.1(53%)
M_3	1.30	1.98	2.41	2.79	NA
M_4	1.00	1.01	1.21	1.62	7.40 (39%)
M_5	1.77	2.62	3.44	6.01	114.78(10%)
M_6	1.68	2.34	2.87	4.37	NA
M_7	2.05	3.38	4.27	7.12	NA
M_8	1.36	1.55	1.71	2.14	7.81(69%)
M_9	1.01	1.02	1.31	1.46	8.32(30%)
M_{10}	1.02	2.56	3.54	5.23	NA
ML_1	1.04	1.15	1.27	1.29	11.78(34%)
ML_2	1.85	2.49	3.38	4.85	NA
ML_3	1.75	2.71	3.25	3.60	7.94(21%)
ML_4	1.72	2.98	3.39	5.80	NA
ML_5	1.05	1.84	2.42	3.42	NA
Bank.	1.08	1.44	1.83	2.66	NA
Doc1.	1.00	1.08	2.21	2.70	I/O Error
Doc2.	1.00	1.01	1.68	1.98	I/O Error
Road T.	1.00	1.01	1.03	1.21	1.41 (100%)

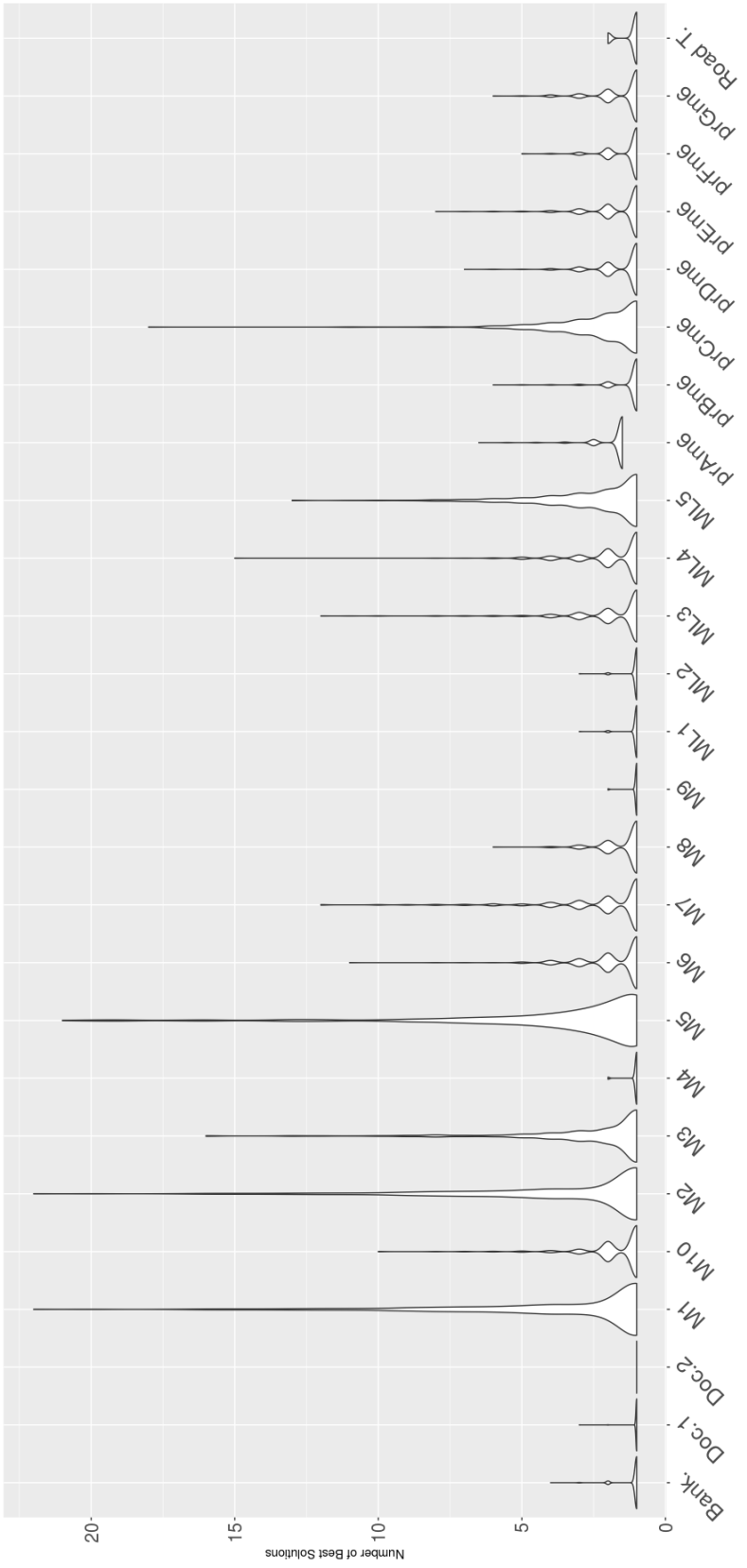


Figure 6.13: Distribution of number of best solutions for 30 iterations.

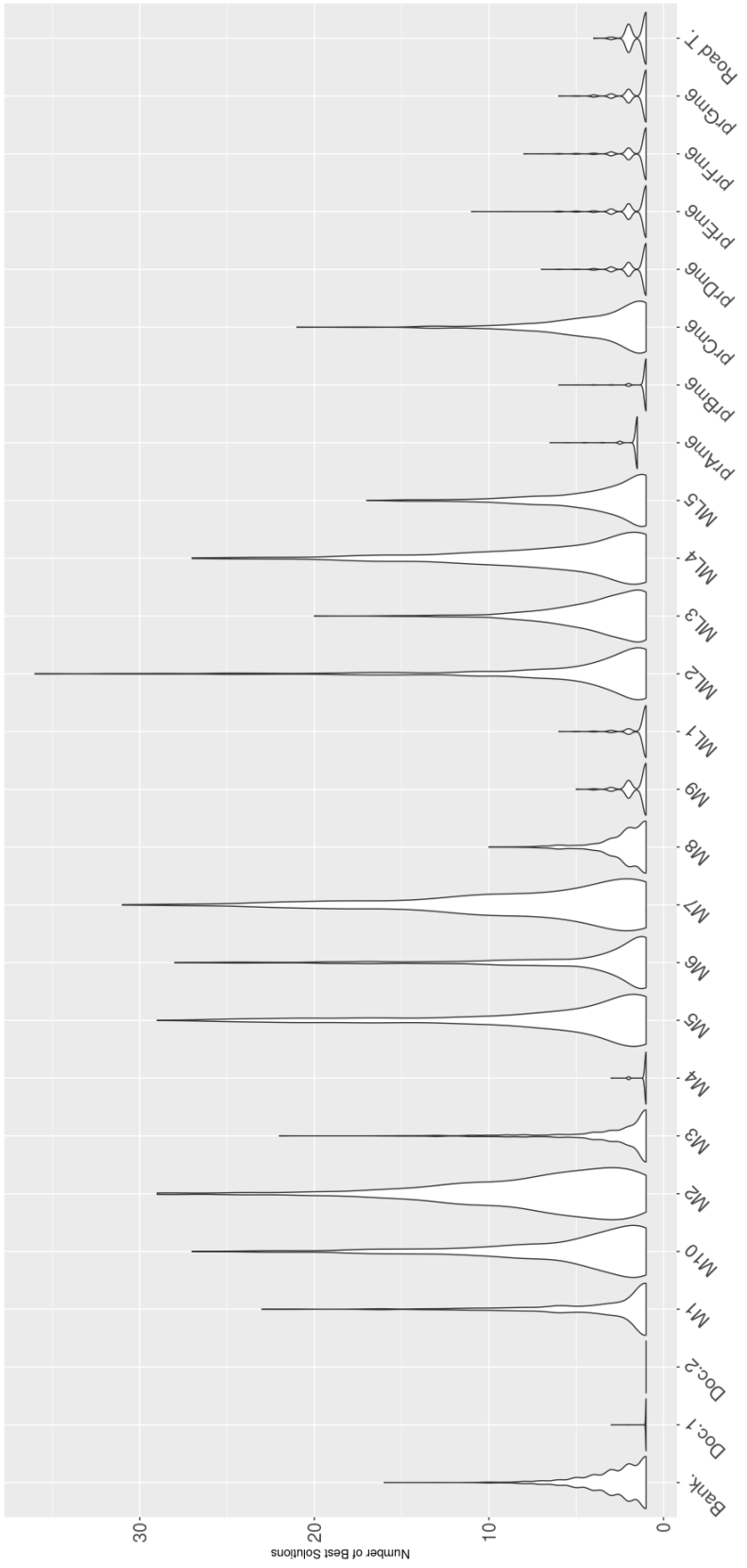


Figure 6.14: Distribution of number of best solutions for 100 iterations.

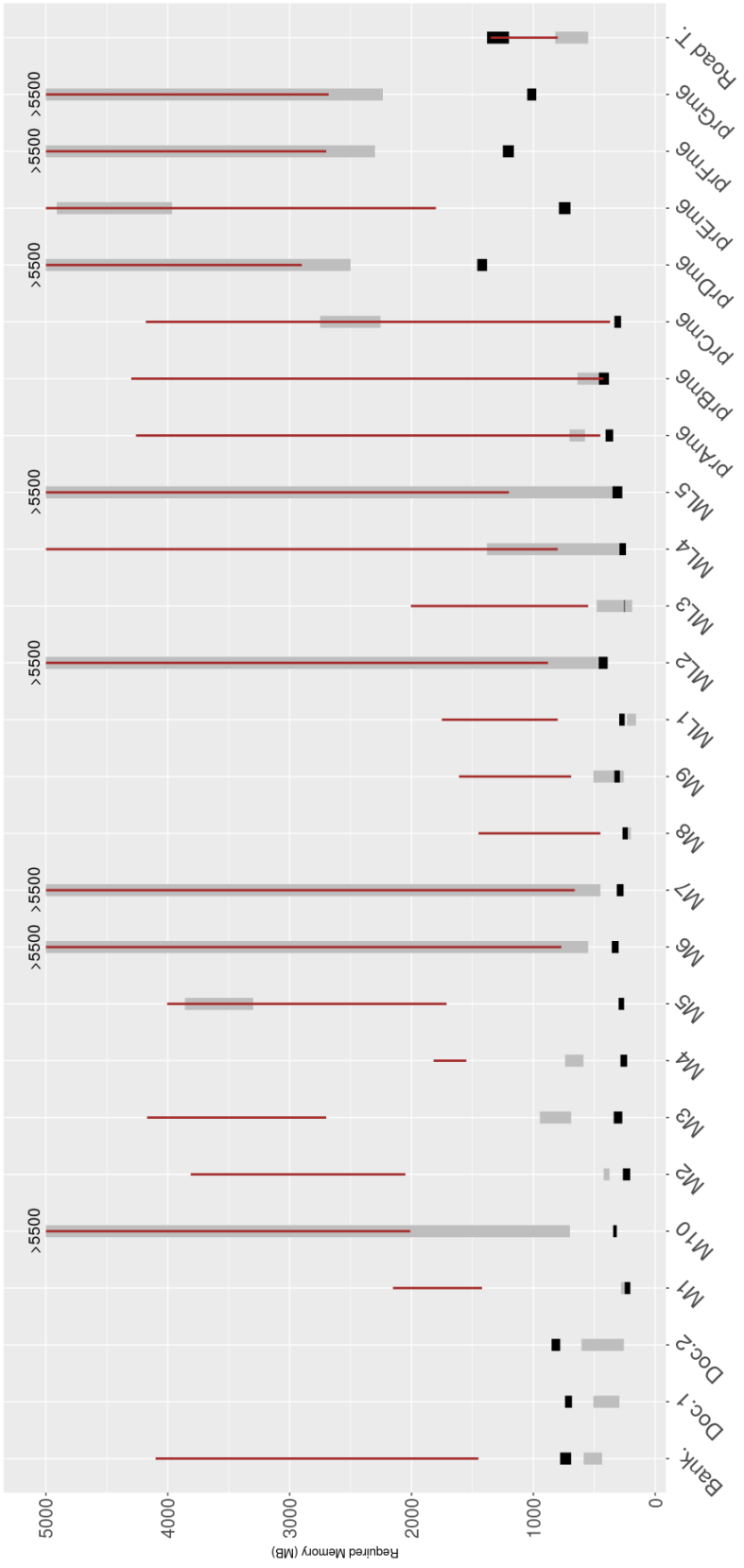


Figure 6.15: Mem. footprint of our approach (black), [5] (gray), and [66] (brown, thin line).

- **Memory Consumption**. The memory footprint of the proposed technique and the ones in [5] and [66], for all the benchmark datasets are represented in Fig. 6.15, using black, gray and brown colors, respectively. It must be stressed that the comparison reported in Fig. 6.15 provides just an indication of the huge difference in terms of memory footprint between the technique of this chapter and the other techniques: for [5], experiments were only done for computing one optimal alignment inevitably, since the implementation for all optimal alignments ran out of memory. In contrast, in Fig. 6.15 we provide the results of our technique and the technique in [66] to compute multiple alignments.

  One can see that the proposed technique requires considerable less memory than the other two techniques. Obviously for small and medium models, the memory footprints are similar. For large models the tendency is inversed: as an example, for *prDm6* the proposed method required around 1.5GB whereas [5] and [66] need more than 5.5GB Notice that the memory footprint of the proposed approach for computing best alignments is bounded through iterations, and is not sensitive to size of the model and length of the observed trace. Also, the required memory for the proposed approach is not sensitive to the labels of transitions i.e., silent or duplicate labels, see $ML_1, \ldots, ML_5$. The other two approaches are more sensitive to the aforementioned factors.

6.5 Outlook

This chapter presents a novel approach to compute several approximation of an optimal alignment. It is based on an evolutionary algorithm, where the memory footprint is guaranteed to be bounded. Tailored genetic operators have been proposed, which help guiding the algorithm through the search space of solutions, and speed up convergence accordingly. The experiments performed on the tool developed witness the quality of obtained alignments, deriving solutions that are close to optimal ones, and which can be improved iteratively. Moreover, the quantity of alignments improves considerably as more genetic iterations are performed. In spite of not having theoretical guarantees on optimality or replayability, the results show that in practice it is always the case that replayable, quasi-optimal or optimal solutions are produced.

Part IV

Reduction and Projection Frameworks

This part of thesis presents new computing styles to alleviate alignment computation problem. Chapter 7 proposes a divide and conquer algorithm to solve very large ILP instances of the method presented in Chapter 3. Following that, in Chapter 8 a new framework for structure reduction of the model and event log will be presented. The main aim of this framework is not alignment computation, but to alleviate the computational burden of the mentioned issue. It can be integrated with all the alignment computation approaches.

Contribution: The work in first chapter of this part, i.e., Chapter 7 alongside with the approach presented in Chapter 3 were published in *International Conference on Business Process Management (BPM 2016)* [74]. The second work in this part, i.e., Chapter 8 was published in *International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016)*-**Best Research Paper Award**. The mentioned approaches have been implemented in Jython (Java-Python) and stand-alone framework ALI [73] respectively.

Chapter 7

Recursive Approach for Large ILP Instances

7.1 Introduction

Chapter 3 presents an ILP approach for alignment computation based on structural theory of Petri nets. Since ILP is NP-hard, casting the problem of computing approximate alignments as the resolution of ILP models is not sufficient for alleviating the complexity of the problem. As the complexity of ILP is dominated by the number of variables and constraints, we present a recursive framework to compute approximate alignments that transforms the initial ILP encoding into several smaller and bounded ILP encodings. This approach reduces drastically both the memory and the CPU time required for computing approximate alignments. Remarkably, it can be applied not only with the ILP encoding used in this chapter, but also in combination with current techniques for computing alignments.

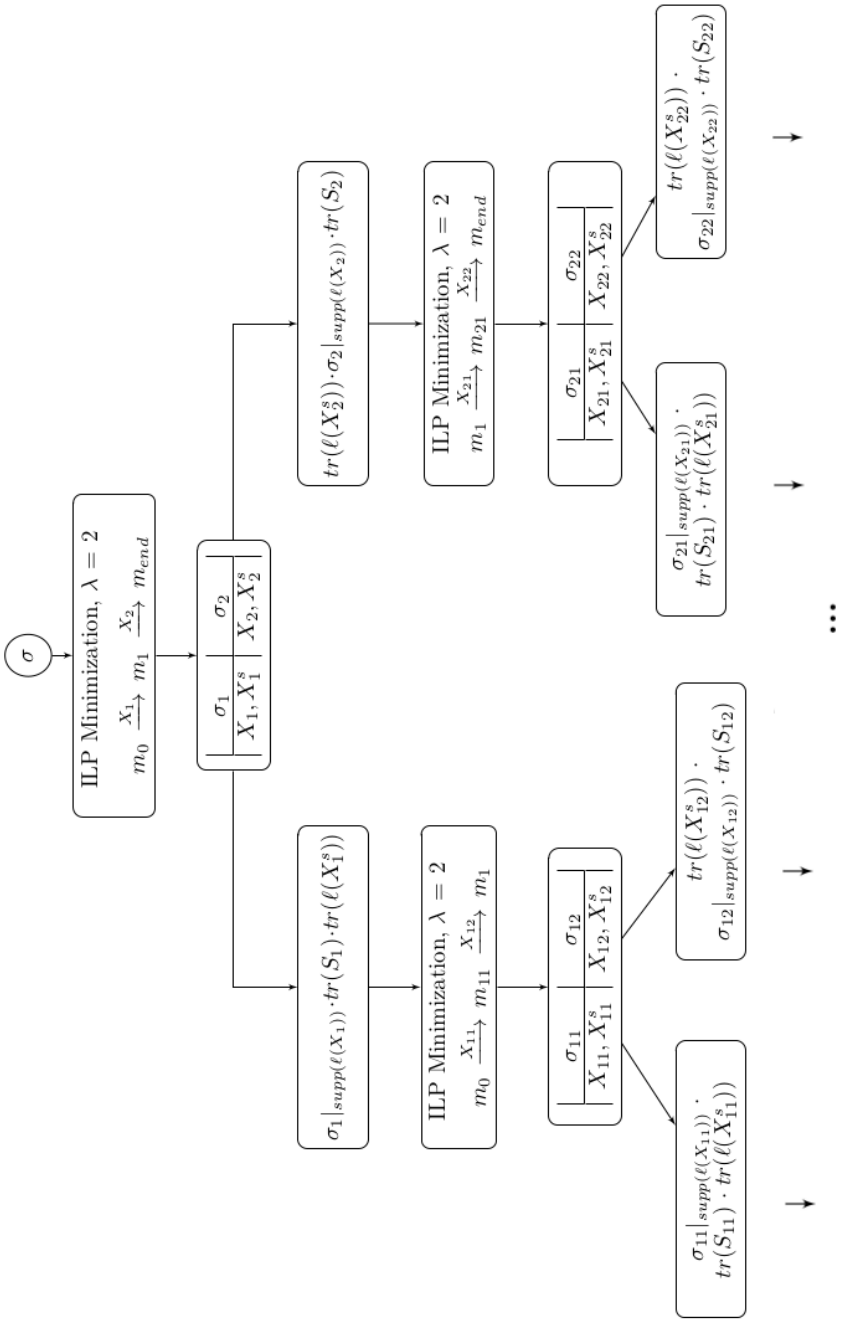


Figure 7.1: Schema of the recursive approach.

7.2 The Recursive Algorithm

Section 3.5 shows how to compute approximate alignments using the structural theory of Petri nets through the marking equation. The complexity of the approach, which is NP-hard, can be measured by the size of the

ILP formulation in the minimization step, in terms of number of variables: given a trace σ and a model with $|T|$ transitions and $|P|$ places, $(|T|+|J|+|P|)\cdot(|\sigma|/\eta)$ variables are needed, where η is the desired granularity and $J = \Sigma \cap \text{supp}(\widehat{\sigma})$. This poses a problem for handling medium/large process models.

In this section we will present a way to fight the aforementioned complexity, by using a recursive strategy that will alleviate significantly the approach presented in the previous section. The first step will be done as in Fig. 3.2, i.e., solving optimization formulation Eq. 3.2. So we will focus on the second step (Ordering), and will assume that σ' is the input sequence for this step[1]. The overall idea is, instead of solving a large ILP instance, solve several small ILP instances that combined represent a feasible solution of the initial problem. Fig. 7.1 illustrates the recursive approach: given a trace σ, on the top level of the recursion a couple of Parikh vectors X_1, X_2 are computed such that $m_{start} \xrightarrow{X_1} m_1 \xrightarrow{X_2} m_{end}$, by using the Ordering ILP strategy of the previous section with granularity $|\sigma|/2$, with $\sigma = \sigma_1\sigma_2$. Some crucial observations can now be made:

1. X_1 and X_2 represent the optimal Parikh vectors for the model to mimic the observed behavior *in two steps*.
2. Elements from X_1 precede elements from X_2, but no knowledge on the orderings within X_1 or within X_2 is known yet.
3. Marking m_1 is the intermediate marking, being the final marking of X_1, and the initial marking of X_2.
4. Elements in $\text{supp}(\ell(X_1)) \cap \text{supp}(\widehat{\sigma_1})$ denote those elements in σ_1 that can be reproduced by the model if one step of size $|\sigma|/2$ was considered [2].
5. Elements in $S_1 = \text{supp}(\ell(X_1)) \setminus \text{supp}(\sigma_1|_{\text{supp}(\ell(X_1))})$, denote the additional transitions in the net that are inserted to compute the final ordering. They will denote skipped "model moves" in the final alignment.
6. Elements in $\text{supp}(X_1^s)$ denote those elements in σ_2 that the model needs to fire in the first part (but they were observed in the second part). They will denote asynchronous "model moves" in the final alignment.
7. 4, 5, and 6 hold symmetrically for X_2, X_2^s and σ_2.

The combination of these observations implies the independence between the computation of an approximate alignment for $\sigma_1|_{\text{supp}(\ell(X_1))}$ ·

[1] For the sake of simplicity and to avoid complicated formulas we rename σ' to σ.

[2] It is worth mentioning that in case of a model with duplicate labels, the mentioned statement might result in unreplayable modeled trace.

$\mathrm{tr}(S_1) \cdot \mathrm{tr}(\ell(X_1^s))$ and $\mathrm{tr}(\ell(X_2^s)) \cdot \sigma_2|_{\mathrm{supp}(\ell(X_2))} \cdot \mathrm{tr}(S_2)$, if the intermediate marking m_1 is used as connecting marking between these two independent problems[3]. This gives rise to the recursion step: each one of these two problems can be recursively divided into two intermediate sequences, e.g., $m_{start} \overset{X_{11}}{\rightarrow} m_{11} \overset{X_{12}}{\rightarrow} m_1$, and $m_1 \overset{X_{21}}{\rightarrow} m_{21} \overset{X_{22}}{\rightarrow} m_{end}$, with $X_1 = X_{11} \cup X_{12}$ and $X_2 = X_{21} \cup X_{22}$. By consecutive recursive calls, more precedence relations are computed, thus progressing towards finding the full step sequence of the model.

Now the complexity analysis of the recursive approach can be measured: at the top level of the recursion one ILP problem consisting of $(|T| + |J_1|) \cdot 2 + |P|$ variables is solved, with $J_1 = T \cap \mathrm{supp}(\widehat{\sigma})$. In the second level, two ILP problems consisting of at most $(|T| + |J_2|) \cdot 2 + |P|$ variables, with $J_2 = \max\{\Sigma \cap (\mathrm{supp}(\widehat{\sigma_1}) \cup \ell(X_1) \cup \ell(X_1^s)), (\Sigma \cap (\mathrm{supp}(\widehat{\sigma_2})) \cup \ell(X_2) \cup \ell(X_2^s))\}$. Hence as long as the recursion goes deeper, the ILP models have less variables. The depth of the recursion is bounded by $\log_2(|\sigma|)$, but in practice we limit the depth in order to solve instances that are small enough.

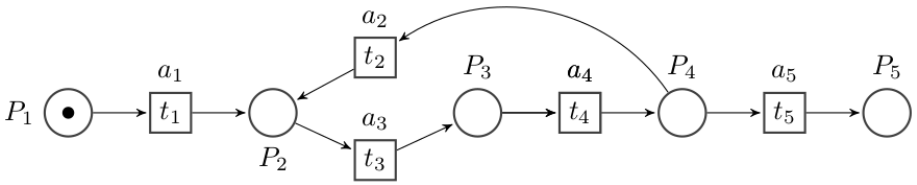


Figure 7.2: Example with loop

Let us show how the method works step by step for an example. Consider the model in Fig. 7.2 and a given non-fitting trace like $\sigma = a_5a_1a_3a_4a_4$ $a_3a_4a_3$. On this trace ILP model (3.2), i.e., seeking an optimal Parikh vector, presented in Section 3.4 will not remove any activity from σ. We then concentrate on the recursive ordering step. First at the top level of Fig. 7.1 the solutions X_1, X_1^s, X_2 and X_2^s will be computed, with $\lambda = 2$.

α_0	$\sigma_1 = a_5a_1a_3a_4$	$\sigma_2 = a_4a_3a_4a_3$
	$X_1 \cup X_1^s = \{t_1, t_3, t_4, t_2\}$	$X_2 \cup X_2^s = \{t_3, t_3, t_4, t_5^s, t_2, t_4\}$

Notice that when seeking for an optimal ordering, t_5 does not appears in X_1 since then its firing will empty the net, and hence it appears in X_2^s (to guarantee reaching the final marking). The intermediate marking computed is $m_1 = \{P_2\}$. Accordingly, $\sigma_1|_{\mathrm{supp}(\ell(X_1))} \cdot \mathrm{tr}(S_1) \cdot \mathrm{tr}(\ell(X_1^s)) = a_1a_3a_4 \cdot a_2 \cdot \emptyset$, and $\sigma_2|_{\mathrm{supp}(\ell(X_2))} \cdot \mathrm{tr}(S_2) \cdot \mathrm{tr}(\ell(X_2^s)) = a_5 \cdot a_4a_3a_4a_3 \cdot a_2$. Let us assume the recursion stops with subtraces of length less than 5, and then the ILP approach (with granularity 1 in this example) is applied. The

[3] Note the different way the traces are obtained, e.g., in the right part $\mathrm{tr}(X_2^s)$ is the leftmost part since it denotes log moves that the model can produce on the left step.

left part will then stop the recursion, providing the optimal approximate alignment:

a_5	a_1	a_3	a_4	$\perp$
$\perp$	t_1	t_3	t_4	t_2

For the subtrace on the right part, i.e., $a_5a_4a_3a_4a_3a_2$ the recursion continues. Applying again the ILP with two steps, with $m_1 = \{P_2\}$ as initial marking, results in the following optimal approximate alignment:

α_1=

$\sigma_{21} = a_5a_4a_3$	$\sigma_{22} = a_4a_3a_2$
$X_{21} \cup X^s_{21} = \{t_3, t_4, t_2\}$	$X_{22} \cup X^s_{22} = \{t_4, t_3, t^s_5\}$

With $m_1 = \{P_2\}$ as intermediate marking. Whenever the recursion goes deeper, transitions are re-arranged accordingly in the solutions computed (e.g., t_2 moves to the left part of α_1, whilst t_5 moves to the right part). The new two subtraces induced from α_1 are $t_4t_3t_2$ and $t_5t_4t_3$. Since the length of both is less than 5, the recursion stops and the ILP model with granularity 1 is applied for each one, resulting in the solutions:

α_{31}=

a_4	a_3	$\perp$
$\perp$	$\{t_3, t_4\}$	t_2

α_{32}=

a_4	a_3	$\perp$
$\perp$	$\{t_3, t_4\}$	t_5

So the final optimal approximate alignment can be computed by concatenating the individual alignments found in preorder traversal:

α=

a_5	a_1	a_3	a_4	$\perp$	a_4	a_3	$\perp$	a_4	a_3	$\perp$
$\perp$	t_1	t_3	t_4	t_2	$\perp$	$\{t_3, t_4\}$	t_2	$\perp$	$\{t_3, t_4\}$	t_5

which represents the step-sequence $\bar{\sigma} = t_1t_3t_4t_2\{t_3, t_4\}t_2\{t_3, t_4\}t_5$ from the model of Fig. 7.1. Informally, the final approximate alignment reports that two activities t_2 were skipped in the observed trace, the ordering of two consecutive pair of events (t_4t_3) was wrong, and transition t_5 was observed in the wrong order. Also, as mentioned in previous sections, the result of proposed method is an approximation to the corresponding optimal alignment, since some moves have non-singleton multisets (e.g., $\{t_3, t_4\}$). For these moves, the exact ordering is not computed although the relative position is known. Cost of the final alignment, i.e., $\Psi(\alpha)$,

according to Def. 20 is sum of the followings:

$$\begin{aligned}
\Psi(\{a_5\}, \bot) &= 1,\\
\Psi(\{a_1\}, \{\ell(t_1)\}) &= 0,\\
\Psi(\{a_3\}, \{\ell(t_3)\}) &= 0,\\
\Psi(\{a_4\}, \{\ell(t_4)\}) &= 0,\\
\Psi(\bot, \{\ell(t_2)\}) &= 1,\\
\Psi(\{a_4\}, \{\bot\}) &= 1,\\
\Psi(\{a_3\}, \{\ell(t_3, t_4)\}) &= 1,\\
\Psi(\bot, \{\ell(t_2)\}) &= 1,\\
\Psi(\{a_4\}, \bot) &= 1,\\
\Psi(\{a_3\}, \{\ell(t_3, t_4)\}) &= 1,\\
\Psi(\bot, \{a_5\}) &= 1
\end{aligned}$$

Thus $\Psi(\alpha) = 8$.

7.3 Experiments

The techniques of this chapter have been implemented in Python as prototype tool that uses Gurobi for ILP resolution[4]. The tool has been evaluated over two different families of examples with different challenges, i.e., (see Appendix A, Tables A.1, A.4). We compare our technique over $\eta = 1$ with the reference three approaches for computing optimal alignments from [5][5]: With or without ILP state space pruning, and the swap+replacement aware[6]. The comparisons are as follows:

[4] The experiments have been done on a desktop computer with Intel Core i7-2.20GHz, and 5GB of RAM. Source code and benchmarks can be provided by contacting the first author.

[5] In spite of using $\eta = 1$, still the objects computed by our technique and the technique from [5] are different, and hence this comparison is only meant to provide an estimation on the speedup/memory/quality one can obtain by opting for approximate alignments.

[6] The plugin "Replay a log on Petri net for conformance analysis" from ProM with parameters "A^* cost-based fitness express with/without ILP and being/not being swap+replacement aware". We instructed the techniques from [5] to compute *one-optimal* alignment.

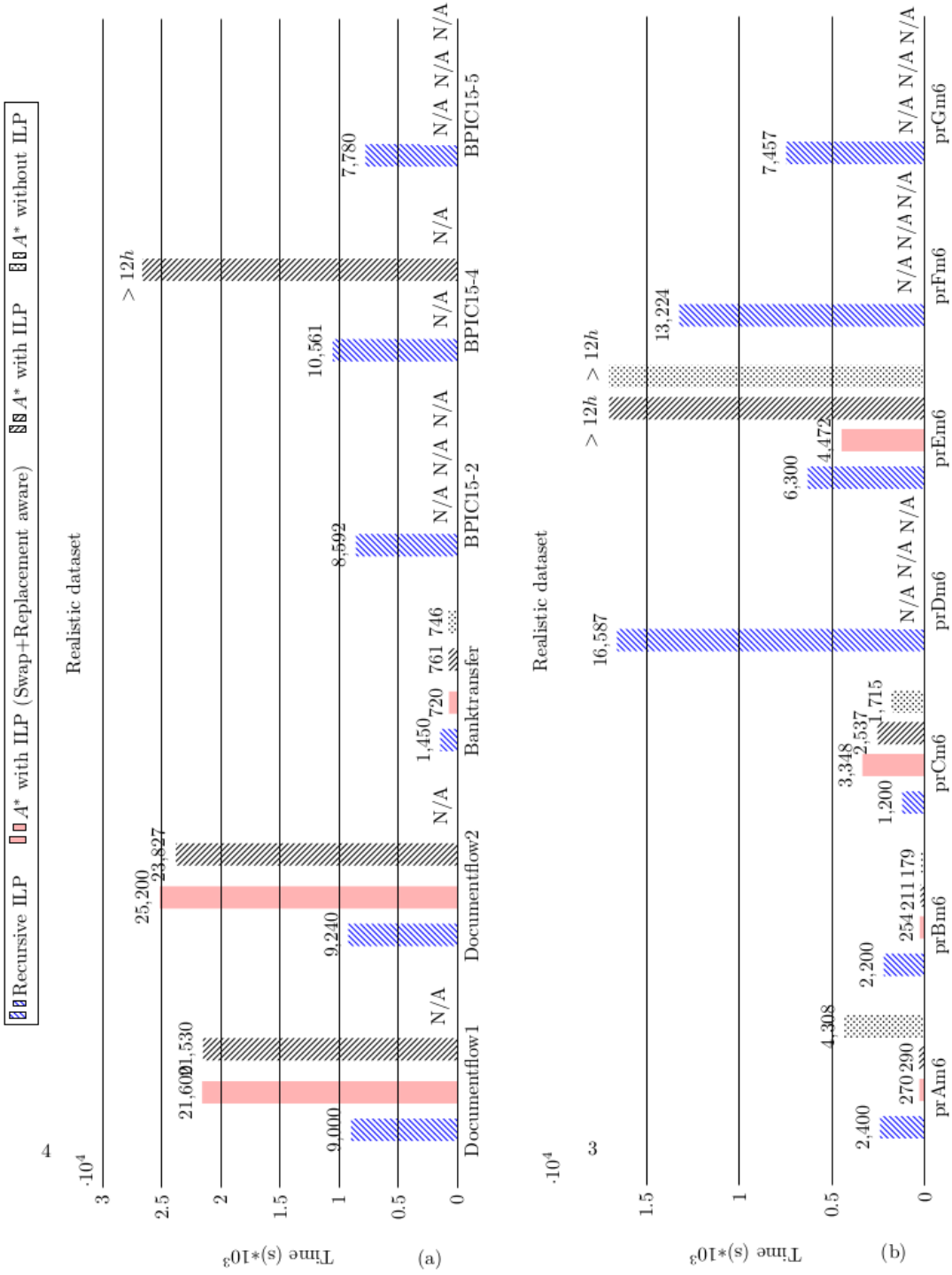


Figure 7.3: Execution times

- **Comparison for Well-Structured and Synthetic Models**: Fig. 7.3 (b) provides the comparison in CPU time for the two families

of approaches for big models presented in Table A.1. One can see that for event logs with many short traces the approach from [5] takes advantage of the optimizations done in the implementation, e.g., caching and similar. Notice that those optimizations can also be implemented in our setting. But clearly, in large models and event logs with many long traces (*prDm6*, *prFm6* and *prGm6*) the three approaches from [5] either provide a solution in more than 12 hours or crash due to memory problems (N/A in the figure), while the recursive technique of this chapter is able to find approximate alignments in a reasonable time. We have monitored the memory usage: our techniques use an order of magnitude less memory than the techniques from [5]. Finally, for these well-structured benchmarks, the approach presented in this technique never found spurious solutions.

- **Comparison for Realistic Benchmarks**: Fig. 7.3 (b) provides the comparison for the realistic examples from Table A.4 presented in Chapter 2. The figure is split into structured and unstructured models, indeed except *Documentflow1, Documentflow2* the rest of realistic models in Table A.4 are structured. For the structural realistic models, the tendency of the previous structured benchmarks is preserved. For the two unstructured benchmarks, the technique of this chapter is able to produce approximate alignments in considerably less time than the family of A^*-based techniques. Moreover, for the benchmarks from the BPI challenge, the A^*-based techniques crashes due to memory problems, whilst our technique again can handle these instances. The memory usage of our technique is again one order of magnitude less than the compared A^*-based techniques, but for the unstructured models spurious solutions were found.

- **Quality of Approximate Alignments**: Table 7.1 reports the evaluation of the quality of the results obtained by the two approaches for the cases where [5] provides a solution. We considered two different comparisons: i) fine-grained comparison between the sequences computed by [5] and the step-sequences of our approach, and ii) coarse-grained comparison between the fitness value of the two approaches. For i), we considered two possibilities: using the *Edit* or *Jaccard* distances[7]. For example for *prAm6* dataset in average less than one operation,i.e., 0.25, is needed to transform one step sequence to the other. In some situations where there are a lot of deviations and the model contains many concurrent transitions, Edit distance is unable to identify very well the similarity of two step sequences, i.e., given a model with many concurrent transitions and a trace with a lot of

[7] Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.

deviations, both approaches provide valid step sequences but their Edit distance is not small. To alleviate this problem, Jaccard distance, which measures dissimilarity between sample sets is used. It considers each step sequence as a set. Although both of metric distances lack some information, they both together can provide more representative information about the similarity of two step sequences. For the first, given a trace σ and a step-sequence $\bar{\gamma}$, we simply take the minimal edit distance between σ and any of the linearizations of $\bar{\gamma}$. For the Jaccard distance, which measures similarities between sets, we considered both objects as sets and used this metric to measure their similarity. In the table, we provide the average of these two metrics per trace, e.g. for prAm6 the two approaches are less than 1 edit operation (0.25) different on average. For measuring ii), the *Root Mean Square Root* (RMSE) over the fitness values provided by both metrics is reported. Overall, one can see that both in fine-grained and coarse-grained comparisons, the approach of this chapter is very close to the optimal solutions computed by [5], specially for well-structured models.

Table 7.1: Quality comparison.

Model/ Case	ED	Jaccard	MSE
prAm6	0.25	0	0.0002
prBm6	0	0	0
prCm6	2.99	0.01	0.0093
prEm6	0	0	0
Banktransfer	4.30	0.04	0.0400
Documentflow	3.16	0.27	0.0310
Documentflow2	3.17	0.29	0.0330

7.4 Outlook

Approximate alignments generalize the notion of alignment by allowing moves to be non-unitary, thus providing a user-defined mechanism to decide the granularity for observing deviations of a model with respect to observed behavior. A novel technique for the computation of approximate alignments has been presented in this chapter, based on a divide-and-conquer strategy that uses ILP models both as splitting criteria and for obtaining partial alignments. The technique has been implemented as a prototype tool and the evaluation shows promising capabilities to handle large instances.

One drawback of this method at the current time is its overhead of multiplying matrices in each step and this is why the state of the art approach had better computation times with small models and short traces. But this problem can be tackled using caching mechanism in order to reduce matrix multiplications and also with using multithreading and pipelining

the preparation time of each trace can be reduced remarkably. On the other hand as explained in section 7.2 the recursive part can be done and implemented in parallel mode on each subtrace as an independent process or thread, where in that case the execution time reduces dramatically and this is the future work of this approach.

Chapter 8

Structure Reduction

8.1 Introduction

This chapter presents a model-based technique for reduction of a process model and observed behavior that both preserves the semantics of the process model and retains the information of the original observed behavior as much as possible. The technique is meant to fight the main problem current approaches for alignment computation have: the complexity both in space and time. In other words, the main purpose of the reduction of a process model and event log presented in this chapter is to alleviate the current computational challenge of computing an alignment, rather than abstracting a process model to capture its essential fragments and hiding details [62]. Therefore given a process model a particular type of relation between transitions which implies *causality* is of interest, and the proposed technique seeks fragments of the process model that easy the computation of the relation. Other types of relation between transitions of the model for the aim of abstraction or consistency verification between process model are presented in [103] which are not suitable for the mentioned challenge.

Reducing the process model and observed behavior cause significant reduction in resources usage for computing alignments, also reducing a process model and observed behavior provide the analyst with an abstract views of them, for example by reducing a process model multiple times the analyst can have an abstract view of the process model which represent the *essence* of its semantic, and by reducing observed behavior consecutive times the analyst can gain an insight into general patterns and what is going on in reality and where they are clustered. As in the previous chapter, we assume that the input models are specified as Petri nets, this is without loss of generality because this formalism can be converted to other formalisms.

The overall idea of this chapter relies on the notion of *indication* between activities of the process model when it is represented as a Petri net. An indication relation between a set of transitions (indicated set) and another transition (indicator) denotes a deterministic causal firing relation in the model, which expresses that the presence in any model's sequence of

the indicator transition requires the presence of the indicated set as well. The notion of indication is inspired from the *reveals* relation from [9] and *co-occurrence* relation in [103]. With the help of *Refined Process Structure Tree* (RPST), we find logically independent parts of a graph (known as *fragment with entry-exit pair* in [63] or the so-called *Single Entry Single Exit* (SESE) in [39]), which are then used to gather indication relations efficiently. These relations dictate which parts of a process model are abstracted as a single, high-level node. Once the model is reduced, the observed trace to align is projected (hence, reduced as well) into the reduced model's alphabet. This way, not only the model but also the trace are reduced, which in turn makes the alignment techniques to be significantly alleviated, specially for well-structured process models where many indication relations may exist. Once alignments are computed, the final step is also an interesting contribution of this chapter: to cast the well-known Needleman-Wunsch algorithm [59] to expand locally each high-level part of the alignment computed, using the indication relation. In must be mentioned that aligning two sequences using this algorithm also used in Chapter 5.

This chapter is organized as follow. The SESE structure and the overal framework will be introduced and detailed in Sect. 8.2 and Sect. 8.3. Following that, in Sect. 8.4, the notion of indication, by which a process model and an observed trace are reduced, is presented. The expansion mechanism for an alignment, that is under the reduced language model is presented in Sect. 8.5. Finally, Sect. 8.6 reports the conducted experiments.

8.2 Single Entry Single Exit (SESE)

The general idea of consistency measure is as follows, given the correspondence relation between the sets of transitions of two WF-nets, all respective transitions of two models are aligned and for each pair of aligned transitions it is checked whether those transitions show the same constraints as defined by the causal behavioural profile. To compute causal behavioural profile efficiently, the presented approach concretises RPST fragments by annotating them with behavioural characteristics. Stated differently, an explicit relation between structural and behavioural characteristics is established.

This section introduces how a WF-net can be partitioned in terms of modular components called Single Entry Single Exit (SESE).

Definition 36 (Mutli-Graph, Subgraph)**.** *A graph is 3-tuple* $\langle V, E, \ell \rangle$*, where* V *and* E *are two disjoint sets of nodes and edges respectively, and* ℓ *is a mapping, which assigns to each edge either an ordered pair of nodes, results in directed graph, or an unordered pair of nodes, results in undirected graph. If a pair of nodes being connected by more than one edge it is called multi-graph. A subgraph can be identified with a pair* $\langle V', E' \rangle$ *where* $V' \subseteq V$ *and* $E' \subseteq E$*. Let* $F \subseteq E$ *represents a set of edges,* $G_F = \langle V_F, F \rangle$

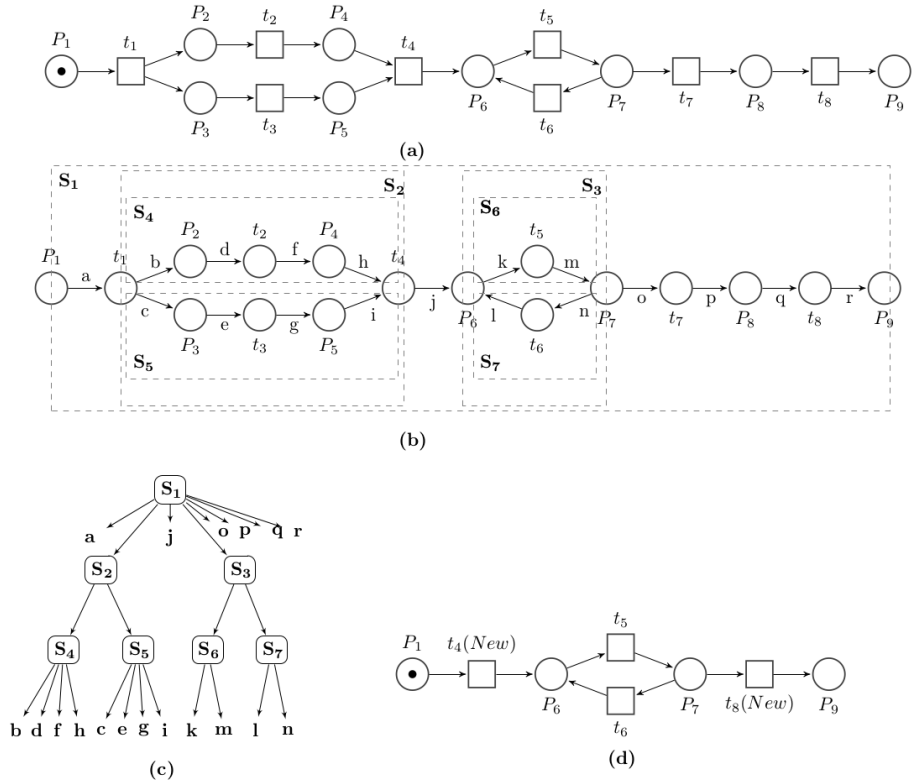


Figure 8.1: (a) WF-net,(b) Workflow graph,(c) RPST, (d) Reduced WF-net

is the subgraph formed by by F if V_F is the smallest set of nodes such that G_F is a subgraph.

A *multi-terminal graph (MTG)* is a directed multi-graph G that has at least one source and at least one sink such that each node lies on a path from some source to some sink. It is *two-terminal graph (TTG)* if it has exactly one source and one sink. So if no distinctions are made between places and transitions of a WF-net, it can be viewed as a TTG or simply called *Workflow graph*. WF-graph of Fig. 8.1(a) is presented in Fig. 8.1(b).

Definition 37 (Boundary Node, Entry, Exit, Fragment)**.** *Let $G = \langle V, E\rangle$ be an MTG and $G_F = \langle V_F, F\rangle$ be a connected subgraph of G. A node in V_F is boundary with respect to G_F if it is connected to nodes in V_F and in $V - V_F$, otherwise it is interior node of G_F. A boundary node u of G_F is an entry of G_F if no incoming edge of u belongs to F or if all outgoing edges of u belong to F. A boundary node v of G_F is an exit node of G_F if no outgoing edge of v belongs to F or if all incoming edges of v belong to F. G_F with one entry and one exit node is called SESE. The entry and exit nodes of G_F is represented by an ordered pair (u, v). F is a fragment of G, if G_F has exactly two boundary nodes, namely one entry*

and one exit. If it contains only one edge it is called trivial. Fragments are represented by a set of edges.

Two SESEs G_{F_1} and G_{F_2} of G are nested if $F_1 \subseteq F_2$ or $F_2 \subseteq F_1$. They are disjoint if $F_1 \cap F_2 = \phi$, otherwise they are *overlapped*. A SESE of G is called *canonical* if it does not overlap with any other SESEs of G. For example in Fig. 8.1(b) all SESEs are canonical, S_2 and S_4 are nested, S_3 and S_2 are disjoint.

Definition 38 (Refined Process Structure Tree)**.** *Let G be an MTG graph, then its Refined Process Structure Tree (RPST) is the set of all canonical SESEs of G. Because canonical fragments are either nested or disjoint, they form a hierarchy.*

In a typical RPST, the parent of a canonical SESE G_F is the smallest canonical SESE that contains G_F. So the leaves of the tree are trivial SESE and the root is the whole graph. Fig. 8.1(c) is the RPST of WF-graph in Fig. 8.1(b), S_1 which is the entire graph is at root and leaves are trivial SESEs which only contain one edge.

8.3 Overall Framework

Given a process model N, represented by a Petri net, and σ as observed behavior, the strategy of this chapter is sketched in Fig. 8.2. We now provide descriptions of each stage.

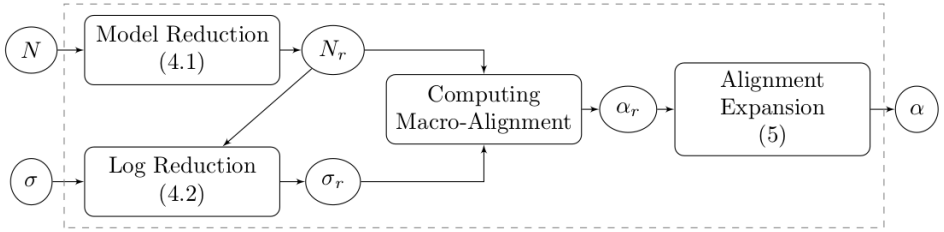


Figure 8.2: Overall framework for boosting the computation of alignments

- *Model Reduction*: N will be reduced based on the notion of *indication* relation which results in N_r. It contains some abstract events representing the indicators of certain indicated sets of transitions. Section 8.4 explains it in detail.
- *Log Reduction*: Using the indication relations computed in the model, σ is projected into the remaining labels in N_r, resulting in σ_r. Section 8.4.2 describes this step.
- *Computing Alignment*: Given N_r and σ_r, approaches like [5] or other methods of alignment computation can be applied to compute alignments. At this point because both N_r and σ_r contain abstract events,

the computed alignment will have them as well. We call it *macro-alignment.*

- *Alignment Expansion*: For each abstract element of a macro-alignment, the modeled and observed indications are confronted. Needleman-Wunsch algorithm [59] is adapted to compute optimal alignments for these abstracted elements. Section 8.5 will be centered on this.

It must be stressed that for the proposed framework, obtaining an optimal alignment is not guaranteed due to the fact that the problem is distributed into several smaller problems for which local optimal alignments are computed. In spite of this, the experimental evaluation presented in Section 8.6 reveals that the results obtained in our framework are often close to the optimal solutions.

8.4 Reduction of Model and Observed Behavior

8.4.1 The Indication Relation

Let us consider the model in Fig. 8.1(a). For any sequence of the model, whenever transition t_4 fires it is clear that transitions t_1, t_3, and t_2 have fired as well or firing of t_8 indicates that t_1, t_5 and t_7 must be happened already. Formally:

Definition 39 (Universal-Indication Relation)**.** *Let $N = \langle P, T, \mathcal{F} \rangle$, $\forall t \in T$, indication is defined as a function, $I(t)$ where, $I : T \rightarrow [P(T)^+]^+$ [1] such that for any sequence $\sigma \in \mathcal{L}(N)$, if $t \in \sigma$ then $I(T) \in \sigma$. If $I(t) = \omega_1\omega_2...\omega_n$, then elements of ω_m precede the elements of ω_n in σ for $1 \leq m < n$. It is called linear if it contains only singleton sets, i.e. $\forall\omega_i \in I(t), |\omega_i| = 1$ otherwise it is non-linear.*

Model reduction can be done through the subclass of universal-indication relation, which is called *flow-indication* relation. Stated formally:

Definition 40 (Flow-Indication Relation)**.** *Given Def. 39, If $I(t) = \omega_1\omega_2...\omega_n$, it represents a flow-indication if and only if, for all consecutive elements ω_i, ω_{i+1}, firing the whole elements of the former enable all elements in the later, exclusively, for $1 \leq i < n$.*

For example in Fig. 8.1(a), $I(t_4) = \{t_1\}\{\{t_2\}, \{t_3\}\}\{t_4\}$ (non-linear), which is a flow-indication as well, and $I(t_8) = \{t_1\}\{t_5\}\{t_7\}\{t_8\}$ (linear), but it is not a flow-indication because firing of t_1 will not enable t_5 exclusively. From now on, because the flow-indication relation is our concern for the remaining parts of the chapter, for the sake of simplicity, by indication we mean flow-indication relation, unless otherwise stated explicitly.

[1] $P(T)$ is powerset of the set of transitions of the model.

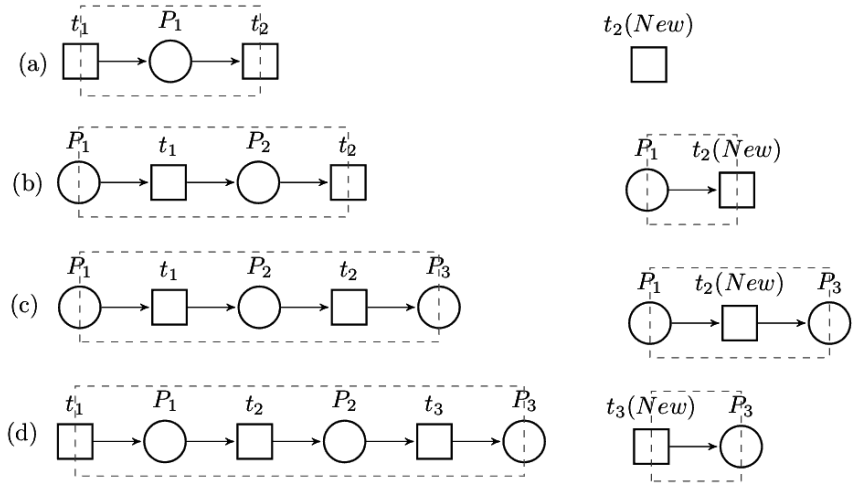


Figure 8.3: Linear SESEs and corresponding reductions.

8.4.1.1 Detecing Flow-Indication Relation through SESE.

SESEs are potential candidates for identifying indication relations inside a WF-net: the exit node of a SESE is the potential indicator of the nodes inside the SESE. Since entry/exit nodes of a SESE can be either place or transitions, SESEs are categorized as (P, P), (P, T), (T, P) or (T, T). In case the SESE is linear, indication relations can be extracted easily and the corresponding SESE is reduced (see Fig. 8.3).

Non-linear cases are decomposed into linear ones such that indication relations can be computed directly on the linear components extracted. After that, the indication relation of the corresponding linear SESEs are computed and they are reduced as well. This procedure should be done with caution to avoid reaching a deadlock situation. Hence a *deadlock-free post-verification* must be done after reduction of these linear parts. Informally, the verification is only needed for particular type of linear SESEs ((T, T)), and consists on validating the property of the SESE after the reduction. Notice the verification is necessary in these cases because, non-linear SESEs may contain linear universal-indications at nested level, which cannot be extracted as flow-indication relations due to choice or loop constructs. For example in Fig. 8.4 (a), (b) t_5 can not be the indicator of transitions in the corresponding SESEs due to choice and loop structures.

Stated differently, the reduction of non-linear SESEs must be done alongside by a deadlock-free post-verification; for instance, Fig. 8.5 shows that in spite of the indication arising from SESE S_2, the net cannot be reduced without changing the language. To put it another way, this reduction will cause a deadlock in the reduced model, and hence must be avoided. Looking at the reduced result in Fig. 8.5 (b), transition $t_5(New)$ never fires because after the reduction it is not enabled since P_4 never gets

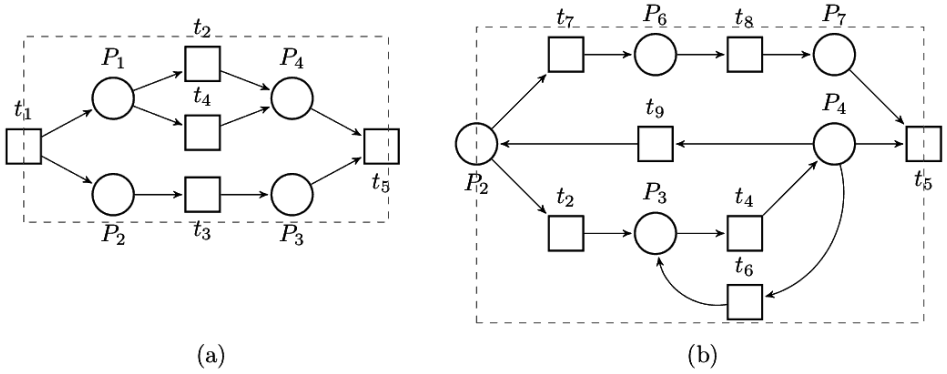


Figure 8.4: (a) Non-Linear (T,T), (b) Non-Linear (P,T)

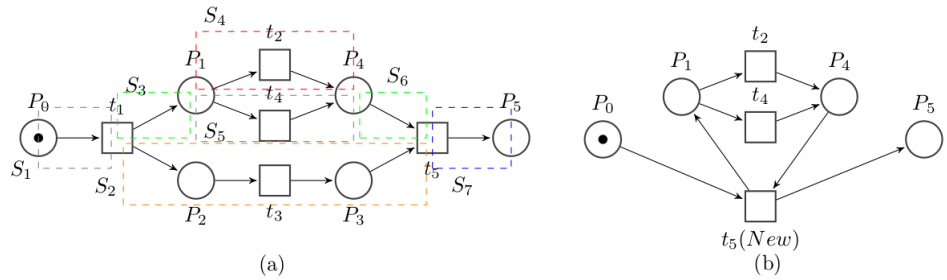


Figure 8.5: Incorrect indication-based reduction: a deadlock is introduced.

marked. To shed more light on the examination of the deadlock-free post verification, more details are stated in the following theorem.

Theorem 6. *Let S be a reduced linear SESE or the combination of other reduced linear SESEs with entry, exit nodes (t_u, t_v) of the (T,T) category. If $OUT(t_u)$ and $IN(t_v)$ represent the set outgoing and incoming arcs of t_u and t_v respectively, then the reduction is deadlock-free if and only if:*

$$a) \quad \forall e \in OUT(t_u), \quad \text{then} \quad e \in S,$$
$$b) \quad \forall e \in IN(t_v), \quad \text{then} \quad e \in S$$

Proof. First of all, assume that the original model before the reduction does not have any deadlock and T_S and $t_{v(New)}$ represent internal transitions of S and the reduced SESE respectively. The proof is presented by contradiction as follow:
Suppose that conditions in Theorem 6 hold and the reduction of S causes deadlock in the system (see deadlock Def. 5 in Chapter 2). Namely, there is a set of places, P_d, which attributes deadlock or in other words $t_{v(New)}$ outputs to one of places in P_d and inputs from one of them. Due to the fact that all transitions in T_S are internal and do not have direct access to any places in P_d, the only incoming and outgoing arcs of $t_{v(New)}$ belong to t_u and t_v respectively. So it can be concluded that once the places in P_d

become unmarked they will always be unmarked and neither t_u nor t_v can place a token in the deadlock, but this contradicts with the assumption that the original model does not have deadlock due to the fact that $IN(t_u)$ and $OUT(t_v)$ remain unchanged before and after reduction. □

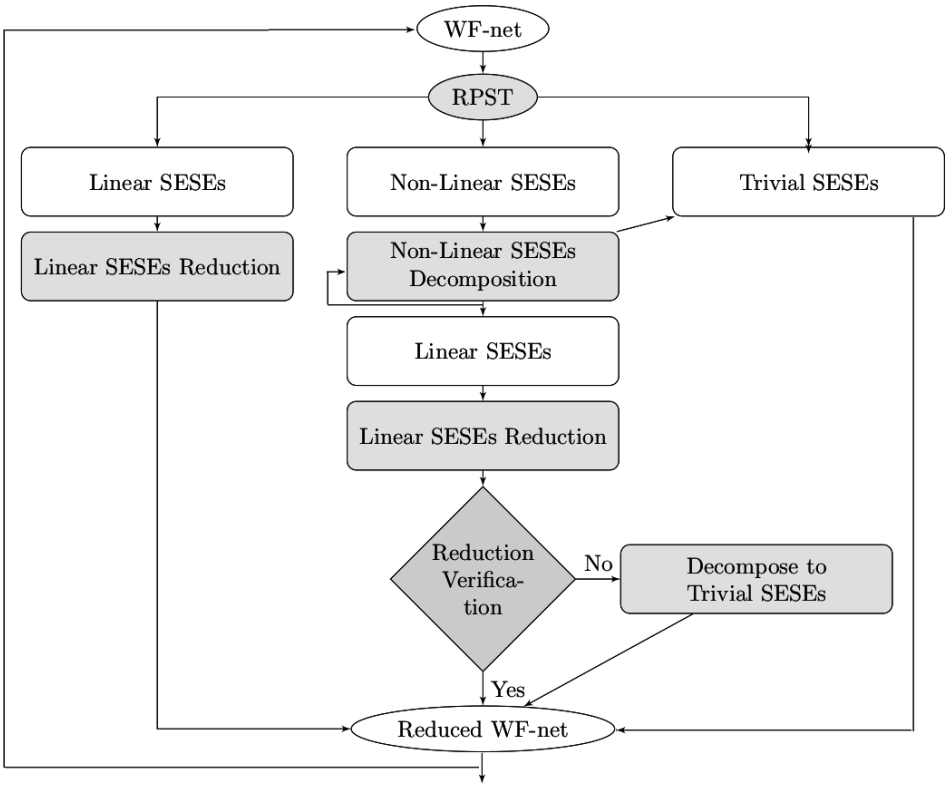


Figure 8.6: Schema for reduction of a WF-net.

The reduction schema is depicted in Fig. 8.6. From the RPST, a top-down approach is applied that searches for indication-based reductions that do preserve the language of the initial model, once the net is expanded back, i.e., the language of the model must be preserved after reduction. Notice that the reduction can be applied more than once till saturation (hence the arc back from the node "Reduced WF-net" to the node "WF-net" in Fig. 8.6).

Fig. 8.7 shows an example (for the sake of simplicity only linear SESEs are shown). Obviously, SESE S_2 is inherently a linear SESE but the rest come from the decomposition of non-linear SESEs. The reduction schema is as follows: Since S_2 is inherently a linear SESE, hence it can be reduced easily according to Fig. 8.3 without any deadlock-free post-verification. The rest of linear SESEs also will be reduced accordingly and the deadlock-free post-verification will be done after each reduction to check that no deadlock arises. One can see all reductions will pass the verification, except for S_7, whose reduction induces a deadlock hence must be excluded from abstraction. Applying the reduction once, results in Fig. 8.7(b). As mentioned earlier, the reduction can be applied more than once until no reduction can

be made. Fig. 8.7(c) is the reduction of the model in Fig. 8.7(b) and it is clear that no more reduction can be made from this model.

8.4.2 Reduction of Observed Behavior

Given a reduced model N_r and σ, we show how to produce σ_r. We will use the reduced model in Fig. 8.7(b) and the trace $\sigma_1 = t_1t_5t_3t_{11}t_{10}t_{21}t_6t_2t_7t_{16}$ $t_{25}t_{19}t_{20}t_{26}$. The reduction of an observed trace is done by the notion of *observed indication*.

Definition 41 (Observed Indication)**.** *Given a reduced model N_r and an observed trace σ, for a model indication $I(t_j)$ the corresponding observed indication is the projection (union of projections) of σ over $I(t_j)$.*

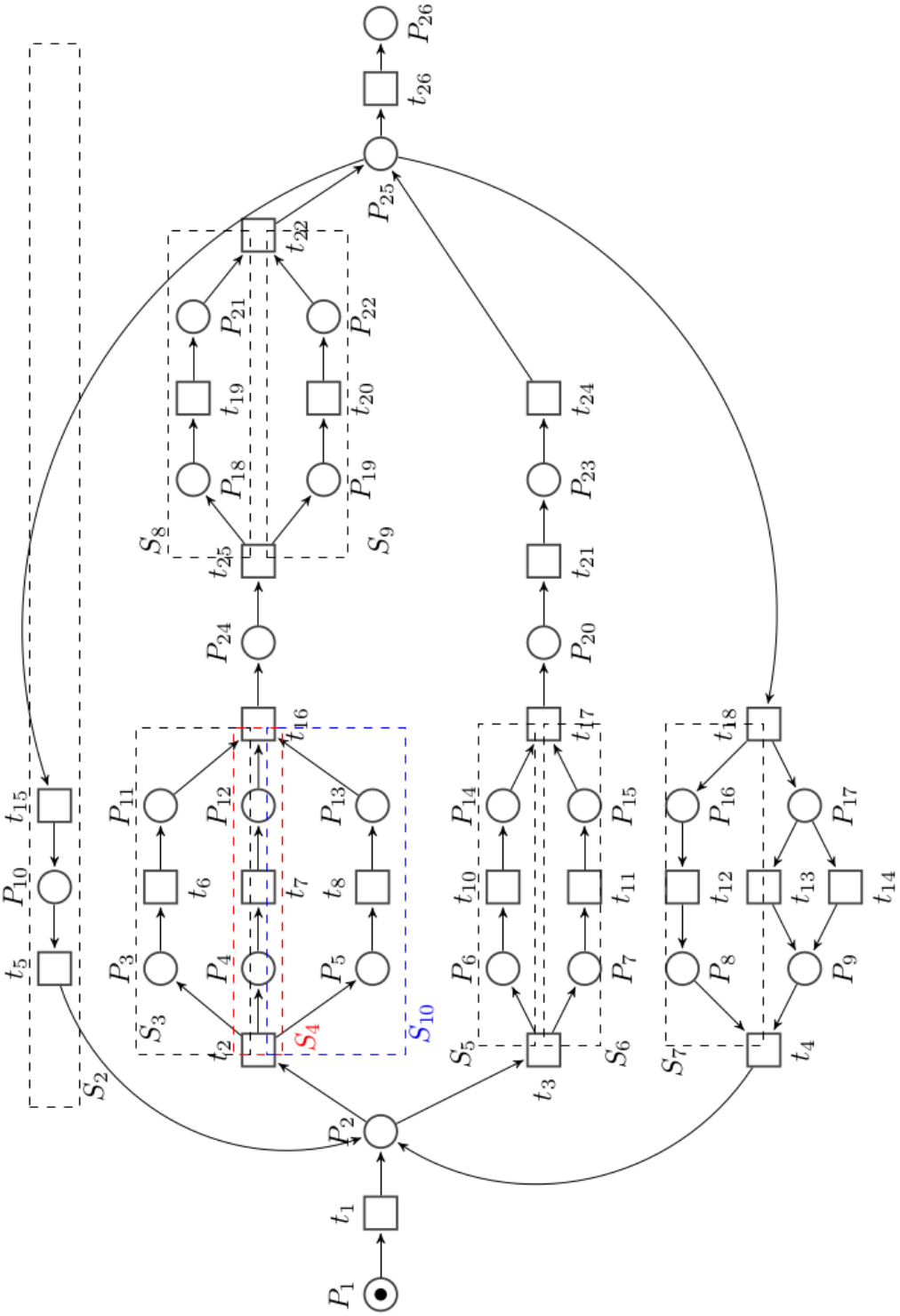
The indication of $t_{5(New)}$ in Fig. 8.7(b) which is linear, equals to $\{t_5\}\{t_{15}\}$. So the observed indication for this abstract node is $\sigma_{1_{\downarrow_{I(t_{5(new)})}}} = t_5$. After computing the observed indication the reduced trace is $t_1t_{5(new)}t_3t_{11}t_{10}t_{21}t_6$ $t_2t_7t_{16}t_{25}$ $t_{19}t_{20}t_{26}$. For $t_{17(New)}$, $I(t_{17(New)}) = \{t_3\}\{\{t_{10}\},\{t_{11}\}\}\{t_{17}\}$, which is non-linear and merged of two linear indications, $I_1(t_{17(New)})$=$\{t_3\}$ $\{t_{10}\}\{t_{17}\}$ and $I_2(t_{17(New)}) = \{t_3\}\{t_{11}\}\{t_{17}\}$. So the projection must be done for each linear indication separately, $\sigma_{1_{\downarrow_{I_1(t_{17(New)})}}} = t_3t_{10}$ and $\sigma_{1_{\downarrow_{I_2(t_{17(New)})}}} = t_3t_{11}$, removing transitions t_3, t_{10}, t_{11} and t_{17} from the current trace (notice that t_{17} does not appear originally, hence it is not projected). Finally, we need to insert $t_{17(New)}$ into the reduced trace; it will be inserted at the position of t_{10}, because the end transition of the abstract node, i.e. t_{17} did not happen in σ, and t_{10} happened last in σ. Therefore the reduced trace so far is $t_1t_{5(new)}t_{17(new)}t_{21}t_6t_2t_7t_{16}t_{25}$ $t_{19}t_{20}t_{26}$. By applying this process for the rest of abstract nodes ($t_{16(New)}$, $t_{22(New)}$), we reach $\sigma_r = t_1t_{5(new)}t_{17(new)}t_{21}t_{16(New)}$ $t_{22(New)}t_{26}$.

8.5 Expansion Through Local Optimal Indication Alignments

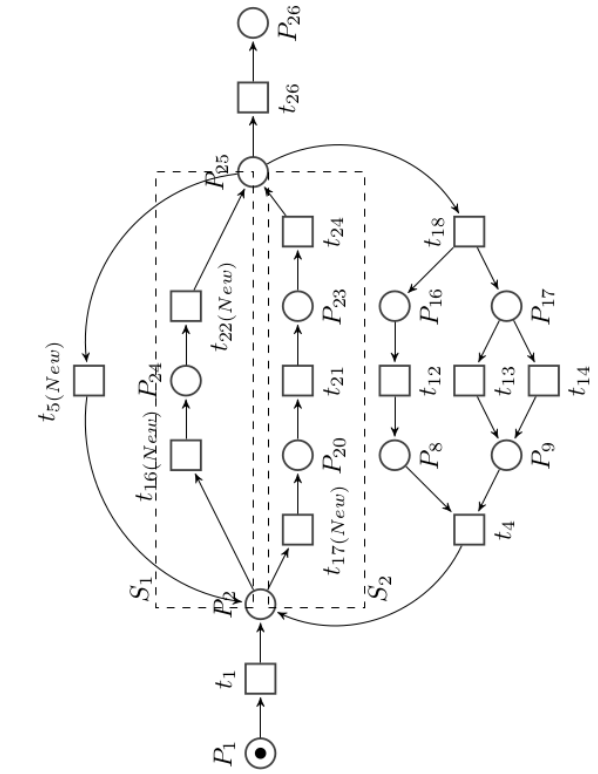
After reducing a given process model and corresponding observed behavior, we can use state of the art approach [5] or other methods for computing alignments to align N_r and σ_r, deriving α_r. For example the following is the macro alignment of $\sigma_{1r} = t_1t_{5(new)}t_{17(new)}t_{21}t_{16(New)}$ $t_{22(New)}t_{26}$ and the model in Fig. 8.7(b) obtained by the approach in [5].

α_r=	t_1	$t_{5(New)}$	$t_{17(New)}$	t_{21}	$\perp$	$\perp$	$t_{16(New)}$	$t_{22(New)}$	t_{26}
	t_1	$\perp$	$t_{17(New)}$	t_{21}	t_{24}	$t_{5(New)}$	$t_{16(New)}$	$t_{22(New)}$	t_{26}

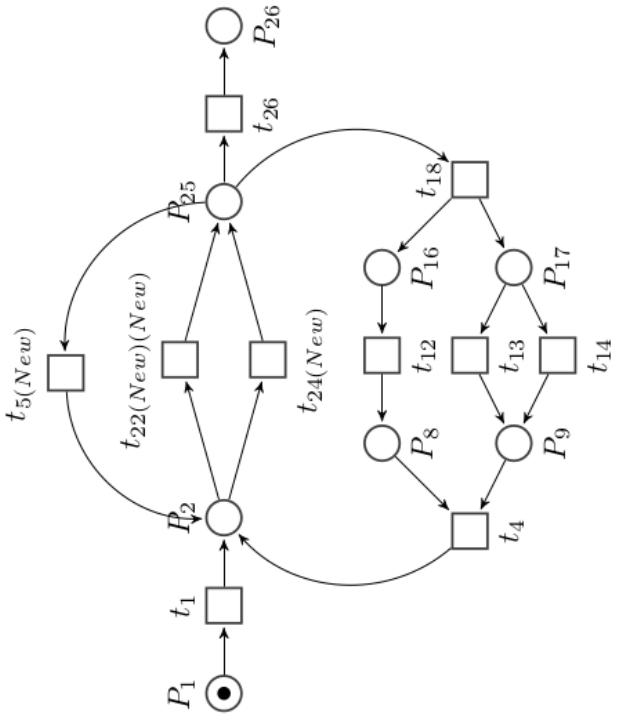
When mapped to linear indications, indication of an abstract node and the corresponding observed indication are both sequence of events; hence for each linear combination of modeled/observed indication, we can adapt the dynamic programming approach from [59] (used in bioinformatics) to


P_1
t_1
P_2
S_2
t_5
P_{10}
t_{15}
S_3
P_3
t_6
P_4
t_7
P_5
t_8
P_{11}
P_{12}
P_{13}
t_2
S_4
t_{16}
S_{10}
P_{24}
S_8
t_{25}
P_{18}
t_{19}
P_{21}
t_{22}
P_{19}
t_{20}
P_{22}
S_9
P_{25}
t_{26}
P_{26}
S_5
P_6
t_{10}
P_7
t_{11}
P_{14}
P_{15}
t_3
t_{17}
S_6
P_{20}
t_{21}
P_{23}
t_{24}
S_7
P_8
t_{12}
P_9
t_{13}
t_{14}
P_{16}
P_{17}
t_4
t_{18}

(a)



(b)



(c)

Figure 8.7: (a) Process model, (b) One-time reduced (c) Two-times reduced.

align two sequences. This approach is fully explained in Chapter 5, Section 5.4. For the sake of simplicity a short description according to this chapter is presented here. As example, we use indication of $t_{17(New)}$ and its observed indication computed in the previous section.

Table 1: Aligning modeled and observed indications

		t_3	t_{11}
	0	-1	-2
t_3	-1	0	-1
t_{11}	-2	-1	0
t_{17}	-3	-2	-1

(a)

		t_3	t_{10}
	0	-1	-2
t_3	-1	0	-1
t_{10}	-2	-1	0
t_{17}	-3	-2	-1

(b)

$\alpha_1 = \begin{array}{|c|c|c|} \hline t_3 & t_{11} & \perp \\ \hline t_3 & t_{11} & t_{17} \\ \hline \end{array}$ $\alpha_2 = \begin{array}{|c|c|c|} \hline t_3 & t_{10} & \perp \\ \hline t_3 & t_{10} & t_{17} \\ \hline \end{array}$

For each linear indication a table will be created, where the first row and column are filled by observed and abstract node indications respectively, as depicted in Table 1(a), 1(b). The second row and second column are initialized with numbers starting from 0,-1,-2,..., they are depicted in yellow color. The task then is to fill the remaining cells as follows:

$$SIM(t_i, t_j) = \text{Maximum}\{SIM(t_{i-1}, t_{j-1}) + s(t_i, t_j), \\ SIM(t_{i-1}, t_j) - 1, \\ SIM(t_i, t_{j-1}) - 1\}$$

Where $SIM(t_i, t_j)$ represents the similarity score between t_i and t_j. $s(t_i, t_j)$ is the substitution score for aligning t_i and t_j, it is 0 when they are equal and -1 otherwise.

The final step in the algorithm is the trace back for the best alignment. In the above mentioned example, one can see the bottom right hand corner in for example Table 1, score as -1. The important point to be noted here is that there may be two or more alignments possible between the two example sequences. The current cell with value -1 has immediate predecessor, where the maximum score obtained is diagonally located and its value is 0. If there are two or more values which points back, suggests that there can be two or more possible alignments. By continuing the trace back step by the above defined method, one would reach to the 0th row, 0th column. Following the above described steps, alignment of two sequences can be found.

Alignments can be represented by a sequence of paired elements, for example $\alpha_1 = (t_3, t_3)(t_{11}, t_{11})(\perp, t_{17})$, $\alpha_2 = (t_3, t_3)(t_{10}, t_{10})(\perp, t_{17})$ and final alignment which represent the non-linear indication is $\alpha = (t_3, t_3)\{(t_{11}, t_{11}), (t_{10}, t_{10})\}(\perp, t_{17})$. This information is booked for each abstract node.

$\alpha=$	t_1	$t_{5(New)}$	t_3	t_{11}	t_{10}	$\perp$	t_{21}	$\perp$	$\perp$	$t_{16(New)}$	$t_{22(New)}$	t_{26}
	t_1	$\perp$	t_3	t_{11}	t_{10}	t_{17}	t_{21}	t_{24}	$t_{5(New)}$	$t_{16(New)}$	$t_{22(New)}$	t_{26}

Table 8.1: Expansion of $t_{17(New)}$

$\alpha=$	t_1	$t_{5(New)}$	t_3	t_{11}	t_{10}	$\perp$	t_{21}	$\perp$	$\perp$	t_6	t_2	t_7	$\perp$	$\perp$	t_{16}	t_{25}	t_{19}	t_{20}	$\perp$	t_{26}
	t_1	$\perp$	t_3	t_{11}	t_{10}	t_{17}	t_{21}	t_{24}	$t_{5(New)}$	$\perp$	t_2	t_7	t_6	t_8	t_{16}	t_{25}	t_{19}	t_{20}	t_{22}	t_{26}

Table 8.2: Expansion of $t_{16(New)}$ and $t_{22(New)}$

$\alpha=$	t_1	t_5	t_3	t_{11}	t_{10}	$\perp$	t_{21}	$\perp$	$\perp$	$\perp$	t_6	t_2	t_7	$\perp$	$\perp$	t_{16}	t_{25}	t_{19}	t_{20}	$\perp$	t_{26}
	t_1	$\perp$	t_3	t_{11}	t_{10}	t_{17}	t_{21}	t_{24}	t_{15}	t_5	$\perp$	t_2	t_7	t_6	t_8	t_{16}	t_{25}	t_{19}	t_{20}	t_{22}	t_{26}

Table 8.3: Expansion of $t_{5(New)}$

After computing local alignments for abstract nodes, we can use them to expand corresponding abstract nodes in a given α_r. The policy of expansion depends on whether the abstract node is in synchronous or asynchronous move. In α_r, $t_{17(New)}$ is in a synchronous move so we can expand it by its local alignment, which results in an alignment depicted in Table 8.1. The same story also happens for $t_{16(New)}$ and $t_{22(New)}$ that are both synchronous moves, the result is shown in Table 8.2. On the other hand $t_{5(New)}$ in α_r is a asynchronous move both on the model and observed trace. The policy of expansion is to expand move on log and move on model independently. To put it in another way, move on log will be expanded using observed indication and move on model will be expanded using the abstract node' indication. The full expansion of α_r is depicted in Table 8.3.

8.6 Experiments and Results

The technique presented in this chapter has been implemented in Python as a prototype tool. The tool has been evaluated over different family of examples with variety of difficulties which are presented in Appendix A. It has been applied alongside with the state of the art techniques for computing alignment [5] (A^*) and the approach presented in Chapter 7.

- **Reduction of Models**. Table 8.4 provides the results of one-time reduction by applying the proposed method to benchmark datasets. Significant reductions are found often. Obviously one can see that the results of reduction are more representative for models without loops like *(prAm6,..,prGm6)* or for models that contain small loops, like (*Banktransfer*).

- **Executable Property of Alignments**. Since the alignment computation based on the approach presented in Chapter 7, i.e., $ILP.R$, may be approximate or the results contain spurious elements, Table 8.5 provides an overview of how many of the computed alignments can be replayed for $ILP.R$ method when combined with the technique of this chapter. Also the corresponding results for state of the art technique in [5] are presented as well. One can see that the expanded alignments provided by A^* were replayed 100% for all datasets.

Table 8.4: Reduced benchmark datasets

Model	$\|P\|$ (Before)	$\|T\|$ (Before)	$\|Arc\|$ (Before)	$\|\sigma\|_{avg}$ (Before)	$\|P\|$ (After)	$\|T\|$ (After)	$\|Arc\|$ (After)	$\|\sigma\|_{avg}$ (After)
prAm6	363	347	846	31	175(52%)	235(32%)	498	22(29%)
prBm6	317	317	752	43	188(40%)	225(29%)	490	33(23%)
prCm6	317	317	752	42	188(40%)	225(29%)	490	33(21%)
prDm6	529	429	1140	248	270(49%)	248(42%)	618	148(40%)
prEm6	277	275	652	98	180(35%)	205(26%)	454	75(23%)
prFm6	362	299	772	240	181(50%)	172(42%)	406	137(42%)
prGm6	357	335	826	143	195(45%)	221(34%)	498	94(34%)
M_1	40	39	92	13	25(37%)	28(28%)	62	9(30%)
M_2	34	34	80	17	26(23%)	28(18%)	64	13(23%)
M_3	108	123	276	37	76(30%)	98(20%)	212	29(21%)
M_4	36	52	106	26	31(14%)	48(8%)	96	23(11%)
M_5	35	33	78	34	27(23%)	27(18%)	62	28(18%)
M_6	69	72	168	53	51(26%)	59(18%)	132	43(19%)
M_7	65	62	148	37	43(34%)	46(26%)	104	28(24%)
M_8	17	15	36	17	6(65%)	7(53%)	14	9(47%)
M_9	47	55	120	44	26(45%)	39(29%)	78	34(23%)
M_{10}	150	146	354	58	91(39%)	105(28%)	236	42(28%)
Bank-transfer	121	114	272	58	61(46%)	72(37%)	152	38(34%)

Table 8.5: Replaying of Computed Step-Sequences

Model	Cases	Replay % (Before) ILP.R	Replay % (After) ILP.R	Replay % (Before) A^*	Replay % (After) A^*
prAm6	1200	100%	100%	100%	100%
prBm6	1200	100%	100%	100%	100%
prCm6	500	100%	100%	100%	100%
prDm6	1200	100%	100%	100%	100%
prEm6	1200	100%	100%	100%	100%
prFm6	1200	100%	100%	100%	100%
prGm6	1200	100%	100%	100%	100%
M_1	500	94.2%	86%	100%	100%
M_2	500	95.4%	86.2%	100%	100%
M_3	500	98%	88.8%	100%	100%
M_4	500	90%	81%	100%	100%
M_5	500	94.8%	95.2%	100%	100%
M_6	500	98.6%	90.8%	100%	100%
M_7	500	97.2%	96%	100%	100%
M_8	500	100%	100%	100%	100%
M_9	500	100%	98.8%	100%	100%
M_{10}	500	100%	99.8%	100%	100%
Bank-transfer	2000	97.25%	88.9%	100%	100%

- **Comparing with Original Alignments**. Table 8.6 reports the evaluation of the quality of the results for both state of the art approaches [5] and the method in Chapter 7 with and without applying the technique of this chapter. Columns ED/Jaccard report the edit/Jaccard distances between the sequences computed, while (Mean Square Error) MSE columns report the mean square error between the corresponding fitness values. Edit distances are often large, but interestingly this has no impact on the fitness, since when expanding abstract nodes although the final position may differ, the model still can replay the obtained sequences very often.

- **Memory Usage**. By one-time reduction, the memory usage[2] of computing alignments using state of the approach [5], is reduced significantly. See Fig. 8.8 which represents the required memory for [5] without and with using the proposed framework respectively. For large models, *prDm6, prFm6, prGm6*, it can only compute align-

[2] Each dataset during its execution was monitored every 0.15 seconds, and the portion of memory occupied by the corresponding process that is held in main memory (RSS) was booked. Based on the gathered data 95% CI was computed.

Table 8.6: Quality of Computed Step-Sequences

Model	ED (A^* vs $EXP.R.A^*$)	Jaccard (A^* vs $EXP.R.A^*$)	MSE (A^* vs $EXP.R.A^*$)	ED (ILP.R vs EXP.R.ILP.R)	Jaccard (ILP.R vs EXP.R.ILP.R)	MSE (ILP.R vs EXP.R.ILP.R)
prAm6	7.49	0	0.065	9.25	0.017	0.00081
prBm6	7.87	0	0	18.31	0	0
prCm6	8.65	0.016	0.005	11.60	0.0019	0.00646
prDm6	NA	NA	NA	93.28	0.0101	0.00041
prEm6	37.14	0	0.02	37	0	0
prFm6	NA	NA	NA	67	0.013	0.0074
prGm6	NA	NA	NA	77	0.011	0.00064
M_1	4	0.085	0.021	4	0.025	0.0165
M_2	6	0.012	0.0193	6	0	0.018
M_3	8	0.046	0.021	5	0.011	0.016
M_4	4	.12	0.028	2	0.015	0.025
M_5	11	0.0022	0.0045	15	0.00024	0.0103
M_6	NA	NA	NA	12	0.0012	0.0088
M_7	NA	NA	NA	15	0.0027	0.019
M_8	4	0.073	0.039	4	0.0078	0.035
M_9	NA	NA	NA	3	0.0044	0.0085
M_{10}	NA	NA	NA	13	0.00038	0.012
Bank-transfer	18	0.031	0.025	13	0.0118	0.0067

ments if applied in combination with the technique of this chapter otherwise it runs out of memory for the machine by which the experiment are done, denoted by (> 5500 MB) in Fig. 8.8. For the approach in Chapter 7, due to the fact that it is based on Integer Linear Programming (ILP), to accentuate the effect of reduction, the evaluation was done based on number of required variables for computing alignments with and without the proposed approach. The results in Table 8.7[3] represent, in average, significant reduction to the number of variables when an ILP instance needs to be solved a given problem.

Table 8.7: Average number of variables for the approach in Chapter 7

Model	$\|Var\|_{avg}$ (Before)	$\|Var\|_{avg}$ (After)	Model	$\|Var\|_{avg}$ (Before)	$\|Var\|_{avg}$ (After)
prAm6	10757	5170 (52%)	M_2	578	364 (37%)
prBm6	13631	7425 (45%)	M_3	4551	2842 (37%)
prCm6	13314	7425 (44%)	M_4	1352	1104 (18%)
prDm6	106392	36704 (65%)	M_5	1122	756 (32%)
prEm6	26950	15375 (43%)	M_6	3816	2537 (33%)
prFm6	71760	23564 (67%)	M_7	2294	1288 (44%)
prGm6	47905	20774 (56%)	M_8	255	63 (75%)
banktransfer	6612	2736 (58%)	M_9	2420	1326 (45%)
M_1	507	252 (50%)	M_{10}	8468	4410 (48%)

- **Computation Time Comparison**.

  Fig. 8.9, 8.10 (a)-(b) report execution times for BPM-2013 and other benchmark datasets for the computation of alignments by techniques in [5] and the one presented in Chapter 7 with and without using the presented technique in this chapter (denoted by *EXP.R.*) respectively. It is evident that A^* approach combined with the proposed method is significantly faster than the other approach in nearly all datasets except (*prGm6, prDm6,* M_6*,* M_{10}). Still A^* approach cannot compute alignments for models M_6 *and* M_{10} even after applying the presented technique, which are denoted by (N/A), and in that case the combination of $ILP.R$ with the presented technique is the best choice.

[3] For a given model with $|T|$ transitions and an event log σ, the required number of variables for the ILP based technique in Chapter 7 is $\Theta(|\sigma| \times |T|)$, totally.

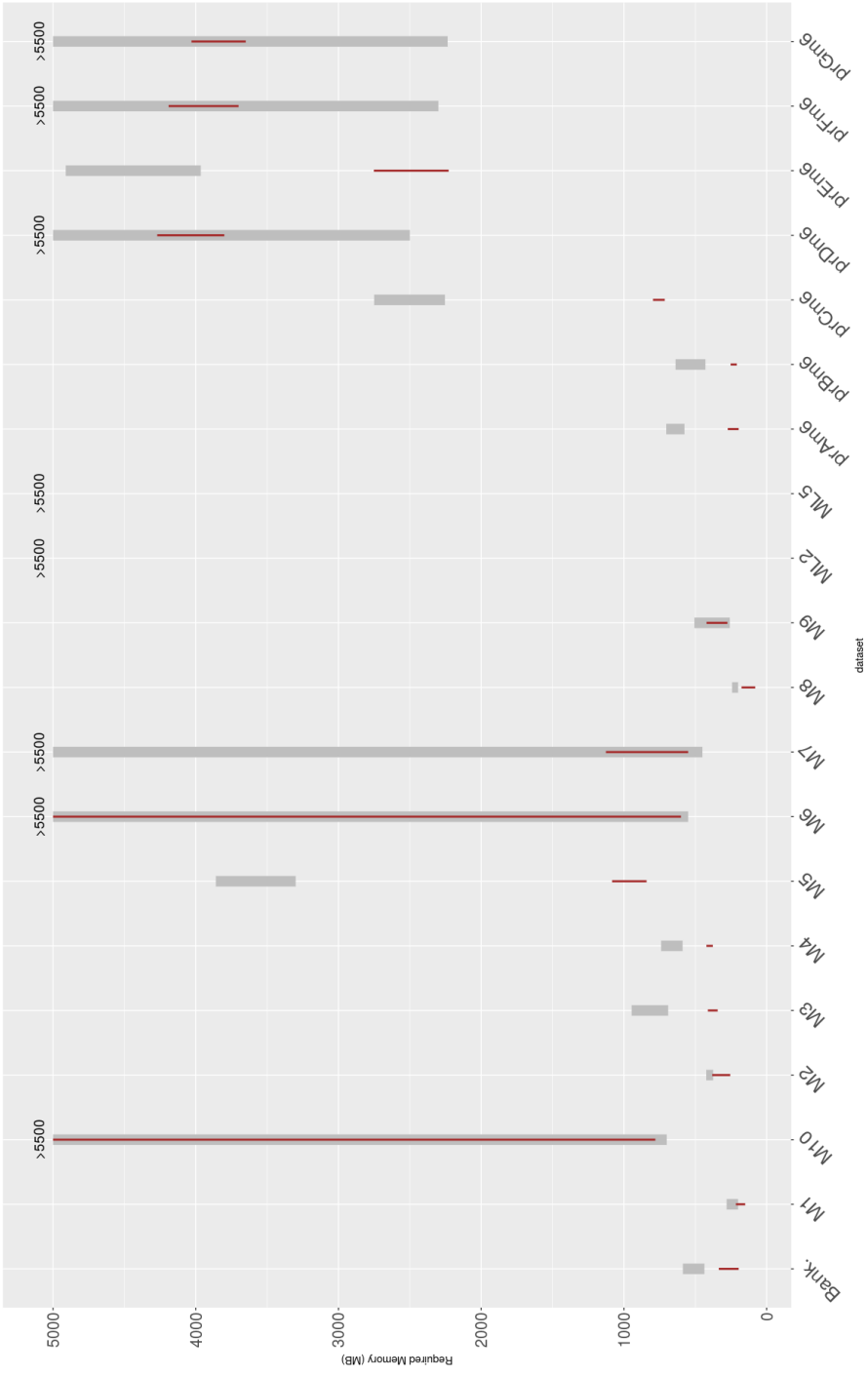


Figure 8.8: Memoery usage of alignment computation before (gray) and after applying technique of this chapter (red)

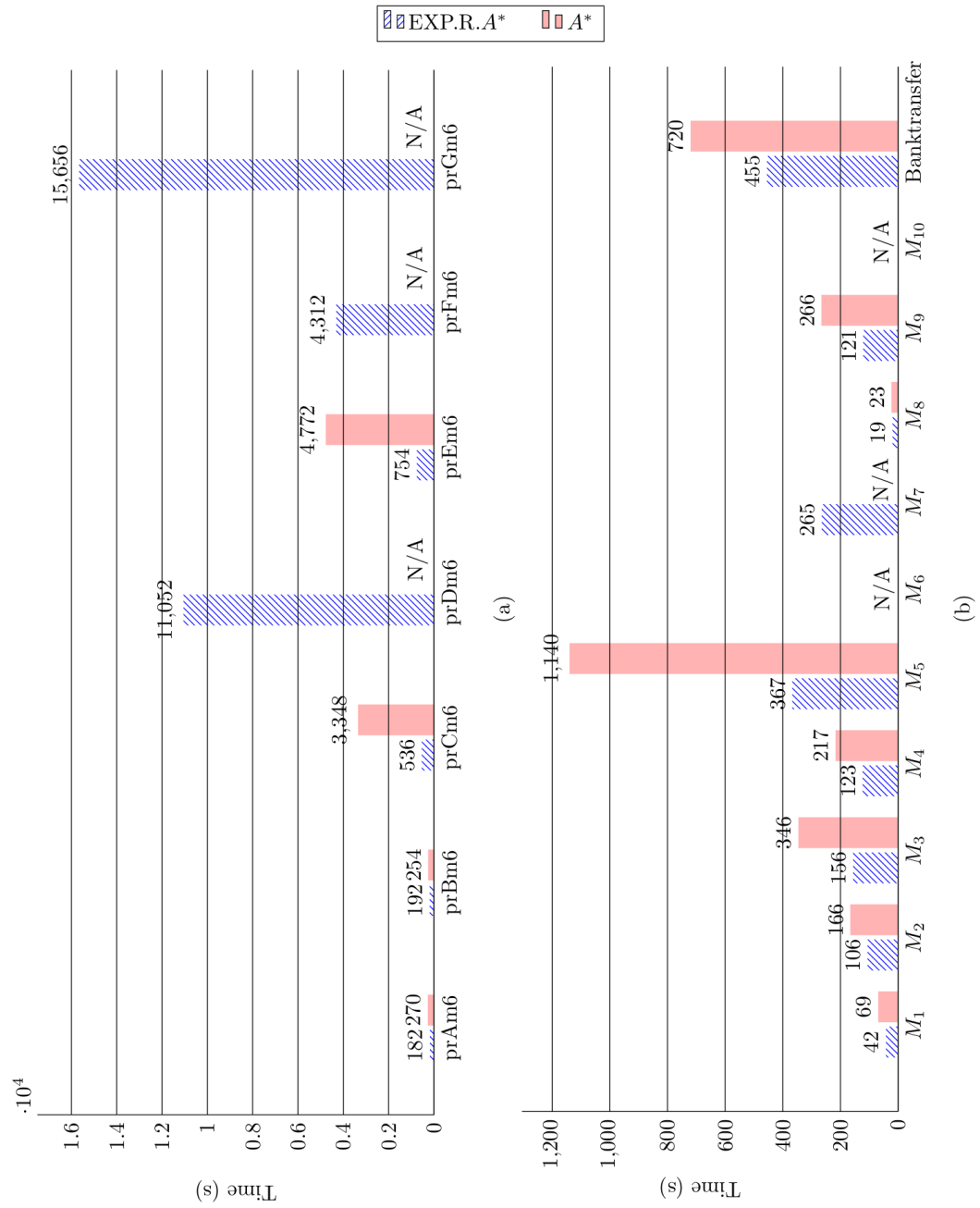


Figure 8.9: (a) BPM-2013 datasets [56], (b) Synthetic datasets

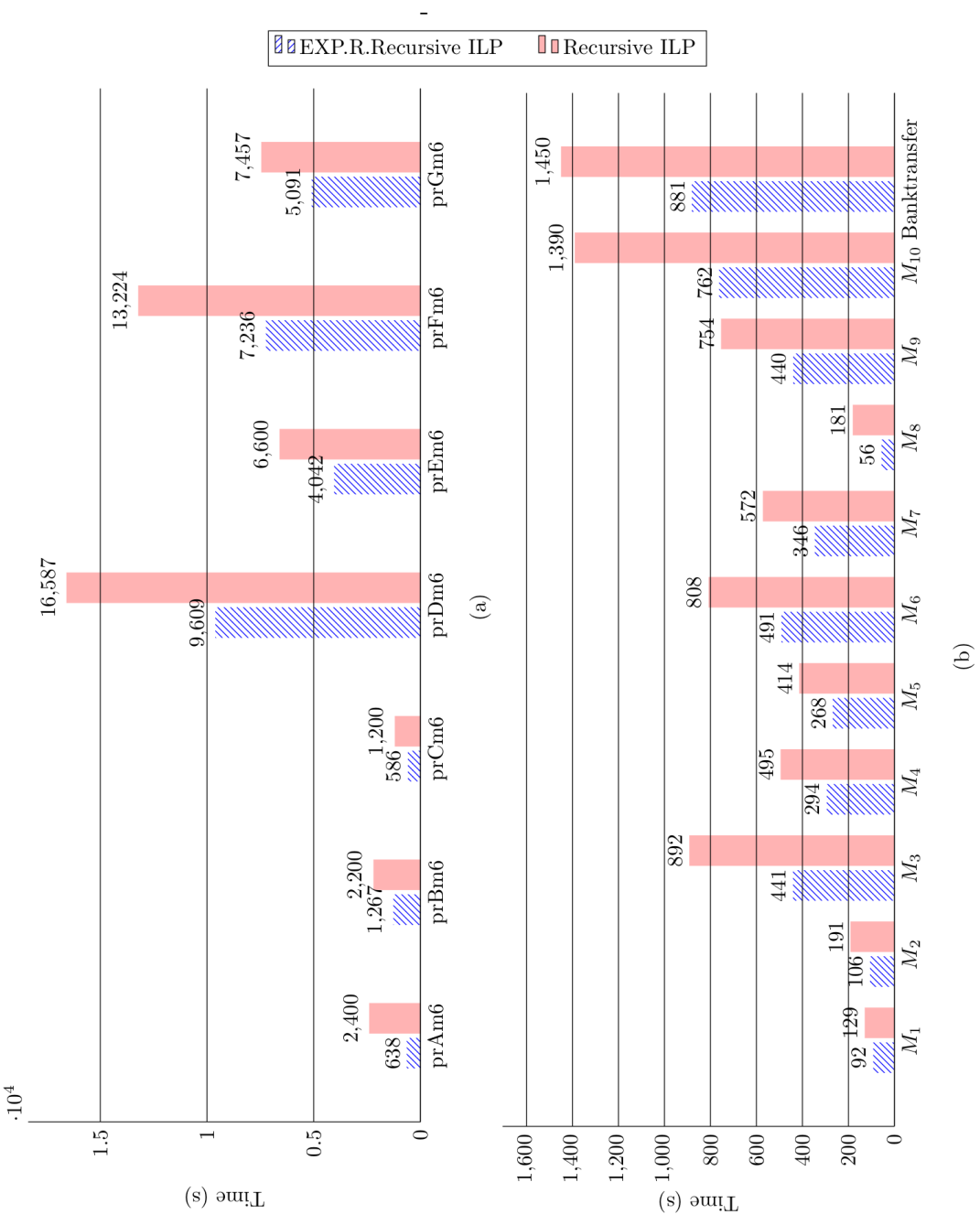


Figure 8.10: (a) BPM-2013 datasets [56], (b) Synthetic datasets

8.7 Outlook

This chapter presented a technique that can be used to significantly alleviate the complexity of computing alignments. The general idea is with preserving language of the given model, making it and corresponding ob-

served trace smaller and more compact. Alignment computation then can be done with these compact representations. Finally the computed alignment will be expanded back in order to have the original representation of the model and observed trace. The technique uses the indication relation to abstract unimportant parts of a process model so that global computation of alignments focus on a reduced instance. The reduced part of computed alignments then will be expanded to represent local deviations as well. Experiments are provided that witness the capability of the technique when used in combination with state-of-the-art approaches for alignment computation.

Part V

Conclusions and Tool Support

Chapter 9

Conclusion

Conformance checking of business process models is a vital task in process mining because it reveals where and to what extend a process model can be lined up to its real execution. Having this knowledge opens a door for further analysis and improvements, such as model repair and more importantly detecting individual deviations, therefore the advantage of having this knowledge is manifold. Mathematical formulation of this issue boils down to the notion of alignment, which details how a model and its event log can be lined up with each other. Alignment computation itself is an optimization problem, in other words given a cost function, the optimal line up between a process model and its footprint is sought. This is a challenging task from computing point of view, since the best devised algorithm has exponential complexity in both memory and time dimensions to the size of a problem.

This research work followed different objectives due to the existing challenges. The main challenge that tackled in all chapters of this thesis is to present ways to alleviate the mentioned existing computation burden. The other objectives which might be a bit overshadowed are both principal and applied like process model summarization, approximate alignment and multiple alignment computation, which have vast applications in numerous domains.

For the main objective, i.e., alleviating optimal alignment computation, this research work tackled it from different angles. The major adopted policy in all proposed techniques is to maintain a balance between three metric dimensions i.e., memory consumption, quality of result and execution time. Each proposed approach, by targeting the main challenge and considering the mentioned dimensions, also takes into account other objectives as outlined earlier. Broadly speaking, all the works done in this thesis can be summarized in three classes:

- The first class adopts classical optimization techniques namely ILP to achieve alignment computation. Specifically, a new optimization paradigm based on ILP, marking equation and structural theory of Petri nets is introduced in Chapter 3. Stated differently, optimal

alignment computation is mapped to another problem which is solving an ILP instance with appropriate constraints. In spite of memory efficiency and providing optimal solution in terms of Parikh vectors, this approach can not deal with big process models since it results in solving a large ILP instance which might be intractable. The novelty of presented approach is to provide an infrastructure for approximate alignment computation, which gives a more coarse perspective of the existing detailed alignment.

Another challenge that this approach faces and can be seen as future work is the existence of spurious solutions which can be more problematic for spaghetti process models. Above that, this approach solves a large ILP instances once, and that constitutes the alignment steps. This raises another challenge and that is, how the number of steps must be estimated or known in advance as a prior knowledge. To cope with the mentioned challenges, Chapter 4 presented alignment computation in an incremental way. More specific, an alignment is computed in two stages. The first stage computes the prefix of an alignment using the ILP method in Chapter 3 and the corresponding tail will be computed by A^* with an heuristic function comes from the first stage. This approach is a trade off between complexity and quality, which means that having an optimal solution might be intractable, thus the quality might be sacrificed to alleviate the existing complexity. Having real solution is guaranteed in this approach. Additionally, computing the tail of an alignment using A^* might requires backtracking which would increase the complexity. Therefore as future work adopting more efficient heuristic functions might be valuable and considered.

Above from these mentioned approaches, the technique in Chapter 7 presented a divide and conquer framework for solving large ILP instances. Indeed, this technique targets the existing challenge in Chapter 3, i.e., intractable property of large ILP instances. This approach by modifying some constraints of the original problem tries to break a large ILP instances into a smaller set of ILP instances, and by solving them in a recursive way in effect provides a solution to the original problem. Although this approach is very efficient in terms of resource usage, it suffers by increasing number of spurious solutions. The root cause of this issue is not difficult to grasp since a smaller ILP that must be solved can potentially inject a spurious solution to the next level of recursion, which can then be propagated consecutively. Therefore as future work one possibility is to manipulate the existing constraints or adding extra constraints to prevent propagation of spurious solutions to the next levels.

- The second class revolves around incorporating modern optimization techniques, like local search and genetic algorithm for alignment com-

putation. It must be mentioned that techniques in this class benefit ILP optimization as well to some extent. Chapter 5 demonstrated a fast and light alignment computation which is based on local search. This approach obtains an initial solution with the help of ILP. Then it is improved by swapping some of its elements and it continues until saturation. This approach works excellent for large process models which contains many parallelization tasks. In contrast, parallel transitions of a process model would be a huge barrier for state of the art technique that explores the corresponding search space. In spite of resource efficiency, low execution time and having close to optimal results, this approach sometimes makes a bad initial solution from where no more improvement can be achieved by swapping. This happens when the parallelization in a model is low, and the root cause is due to the adopted heuristic whose policy makes an initial solution that is far from the desired solution. On these situations, swapping operations will no longer be beneficial. Therefore, one possibility of future work would be devising better and more stable heuristic functions.

Apart from that, to overcome the mentioned challenge, Chapter 6 presented an evolutionary approach for alignment computation. It computes an alignment in two steps, the fist one is similar to the technique in Chapter 5, but instead of improving the initial solution, it generates a set of candidate solutions (chromosomes) by permuting the elements of an initial solution. Having a set of candidate solutions is twofold, firstly, the probability of getting stuck in local optima decreases and, secondly, due to the nature of GA it might come up with more than one unique solution. Also, domain specific operators have been devised to inject the knowledge of alignment computation to GA. Having domain specific operators cause first to speed up the algorithm, and second to prune the respective search space. It must be noted that due to lack of convexity of the respective search space and stochastic nature of GAs, there is no guarantee on having optimal solutions, but by developing efficient and domain dependent operators the probability of having desired solutions can be increased. Despite of being slow execution time which is the nature of GA, this approach is designed to be memory bounded, namely, the number of candidate solution is fixed from time to time, and only candidate solutions are replaced with improved ones.

- The third class of alignment computation in this research is a general framework which can be integrated with all alignment computation methods. It must be mentioned that for all methods of alignment computation, the size of an input problem, i.e., process model and event log, as a parameter affects resources consumption. This effect varies for different techniques, and its consequences over methods in this thesis is not to the extend it does over state of the art technique.

The main objective of this framework is to summarize the existing process model and event log by preserving the respective semantics. To achieve this goal, Chapter 8 proposed an approach in which a process model is decomposed into smaller parts that can be regarded totally as an atomic unit. Next, the event log is projected into these new units to be in accord with the new summarized model. Stated differently, it provides a coarse perspective of existing model and corresponding event log. Although alignment computation in summarized dimension or language is less severe than the original one, the result must be expanded to the initial dimension, which sometimes sacrifices optimality of the obtained solutions. The loss varies across different alignment computation methods under consideration. Another contribution of this framework lays on the fact that this model reduction provides a big picture of the existing WF-net in a hierarchical way, which would be useful for other tasks than alignment computation, like model understanding. Also it must be mentioned that the underlying technique for model summarization works well with well-structured models, and it is less applicable for Spaghetti ones which can be alleviated by incorporating other decomposition techniques.

At the end, it must be said that all the techniques in this thesis were implemented in Python from scratch, and the source codes are available publicly. More importantly, a framework which provides a GUI and having all these techniques was developed which is called Alignment Large Instances (ALI) which you can find its manual in the next chapter.

Chapter 10

Tool Support

10.1 Introduction

ALI is a framework of alignment computation for a given Process Model and an Event Log. It has been developed using Python 2.7. The standalone files for both Linux based and Microsoft Windows Operating systems can be downloaded from https://www.cs.upc.edu/~taymouri/tool.html. It has been developed to work with all or some CPUs of the system under which it works. Also, since many techniques of this thesis are based on ILP, therefore, ALI is integrated to work with open-source LP solvers like *Pulp*[1] as well as commercial LP solvers like Gurobi [2].

10.2 Installation

ALI automatically installs the required Python libraries (if you don't have them on your machine) for the fist time you run it. It installs the packages via *pip*[3]. So be sure to have installed it alongside with Python 2.7 on your machine.

- **Linux:** Please download the corresponding distribution for Linux OS from https://www.cs.upc.edu/~taymouri/tool and unzip it somewhere on your local disk. Open the terminal and change your current directory to where you downloaded the distribution, i.e., "cd /path to the distribution folder /Linux ". After that, in the terminal make the shell script executable via command "chmod +x AlI.sh". Finally, just type "./ALI.sh" and press Enter. For the first time it installs the required packaged and then ALI will be ready to use.
- **Windows:** Please download the corresponding distribution for Microsoft Windows OS from https://www.cs.upc.edu/~taymouri/tool

[1] https://pypi.python.org/pypi/PuLP
[2] http://www.gurobi.com/index
[3] https://pypi.python.org/pypi/pip

and unzip it somewhere on your local disk and then open it. To run ALI just click on "ALI.bat".

10.3 Importing Model and Log

ALI accepts Process models in *PNML* format which is a Petri Net Markup Language (PNML) and it is a proposal of an XML-based interchange format for Petri nets. Also Event Logs must be presented in *Extensible Event Stream (XES)* format [4].

To import the model and event log files click on corresponding Location buttons, select your model and even log. To specify the destination (where the selected algorithm puts the final results) follow the same rule by the Location button. The following picture, shows a snapshot of the tool when it is fed by a model and an even log.

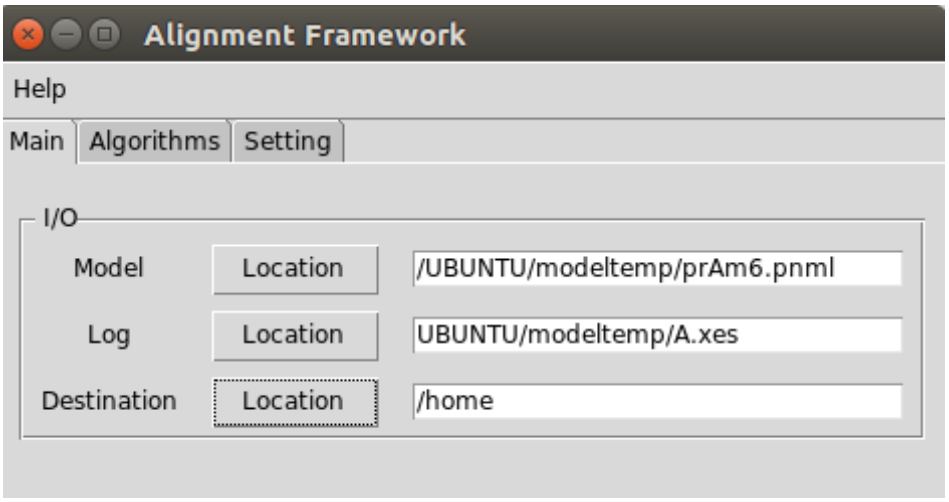


Figure 10.1: I/O Operations

10.4 Algorithms

This tab of ALI contains the related developed algorithms (It will be updated with more algorithms).

10.4.1 ILPSDP

This algorithm is called *Integer Linear Programming-Sequential Dynamic Programming (ILPSDP)* and presented in-depth in Chapter 5.

[4] http://www.xes-standard.org/

10.4.1.1 Inputs

- This technique only accepts WF-nets. This approach of alignment computation first with the use of ILP finds a Parikh vector which is as similar as possible to the Parikh vector of observed trace (this similarity can be adjusted by *Delta* parameter, the default value is -5. The greater absolute value of this parameter the more penalty the skipped transitions do receive).
- Next, it tries to execute the computed Parikh vector (from the initial marking to the final marking; *end* parameter denotes the final place of the WF-net) to obtain an executable sequence.
- When you specified the mentioned issues, you can click on Run to execute the approach. The following is a snapshot of ALI for the mentioned issues.

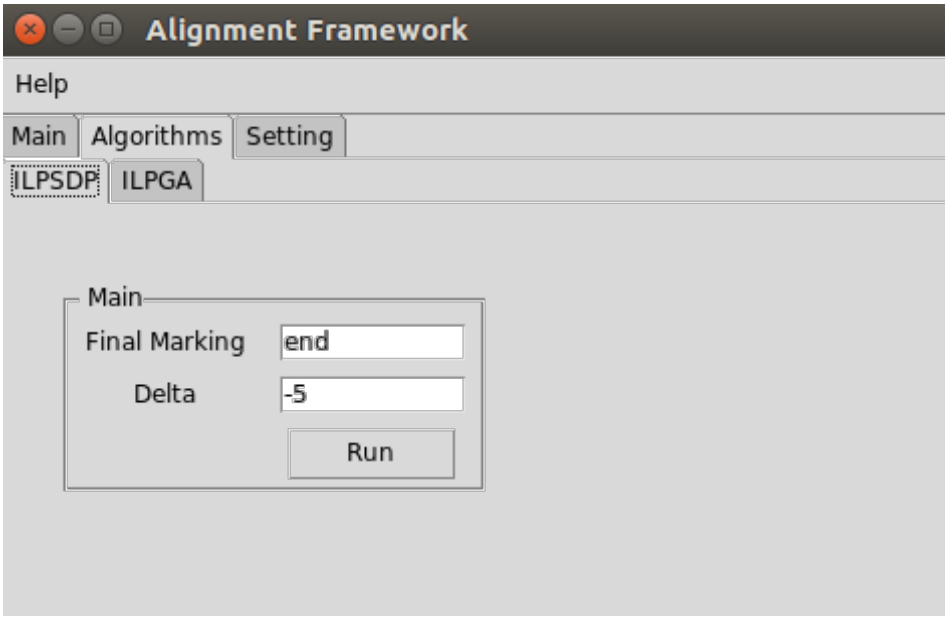
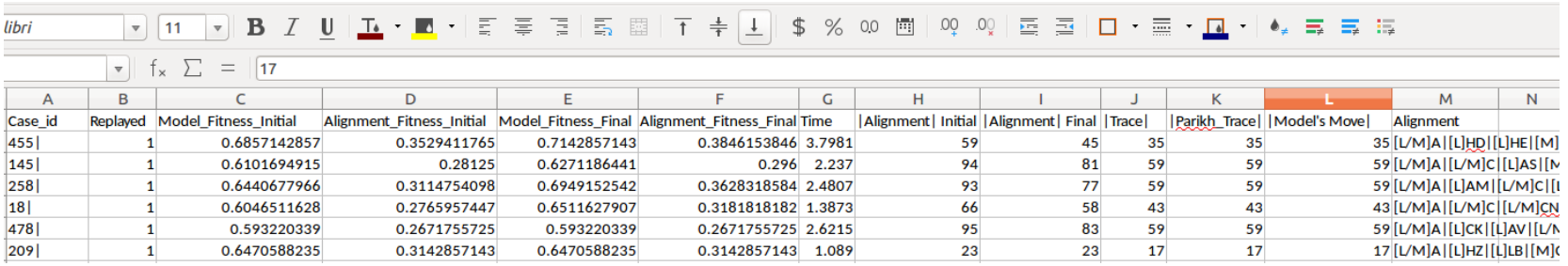


Figure 10.2: ILPSDP settings

10.4.1.2 Outputs

ILPSDP algorithm, provides an Excel file which has different columns for each row, i.e., an observed trace, as shown in the following snapshot. Description of each column are as follows:

- *Case-id*: It denotes the id of each case in XES file. Sometimes, some cases are the same hence only one row is considered for all of them.
- *Replayed*: This column denotes whether the obtained alignment is executable or no. The value "1" represents an executable alignment



libri | 11 | 17

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Case_id	Replayed	Model_Fitness_Initial	Alignment_Fitness_Initial	Model_Fitness_Final	Alignment_Fitness_Final	Time	|Alignment| Initial	|Alignment| Final	|Trace|	|Parikh_Trace|	|Model's Move|	Alignment	
455	1	0.6857142857	0.3529411765	0.7142857143	0.3846153846	3.7981	59	45	35	35	35	[L/M]A | [L]HD | [L]HE | [M]	
145	1	0.6101694915	0.28125	0.6271186441	0.296	2.237	94	81	59	59	59	[L/M]A | [L/M]C | [L]AS | [M	
258	1	0.6440677966	0.3114754098	0.6949152542	0.3628318584	2.4807	93	77	59	59	59	[L/M]A | [L]AM | [L/M]C | [	
18	1	0.6046511628	0.2765957447	0.6511627907	0.3181818182	1.3873	66	58	43	43	43	[L/M]A | [L/M]C | [L/M]CN	
478	1	0.593220339	0.2671755725	0.593220339	0.2671755725	2.6215	95	83	59	59	59	[L/M]A | [L]CK | [L]AV | [L/M	
209	1	0.6470588235	0.3142857143	0.6470588235	0.3142857143	1.089	23	23	17	17	17	[L/M]A | [L]HZ | [L]LB | [M](	

Figure 10.3: Output of ILPSDP algorithm in an Excel file

and "0" otherwise.

- *Model Fitness Initial*: It denotes the fitness of the modeled trace when the base alignment is obtained.
- *Alignment Fitness Initial*: It denotes the fitness of initial or base alignment.
- *Model Fitness Final*: It denotes the final fitness of the modeled trace when by filliping elements of initial alignment no improvement of fitness can be obtained.
- *Alignment Fitness Final*: It denotes the fitness of final alignment.
- *Time*: This column denotes the execution time of obtaining the final alignment for a given observed trace.
- $|Alignment|$ *Initial*: It denotes the total number of moves (synchronous + asynchronous) of the initial alignment.
- $|Alignment|$ *Final*: It denotes the total number of moves (synchronous + asynchronous) of the final alignment.
- $|Trace|$: It denotes the length of the observed trace (the number of events).
- $|ParikhTrace|$: It denotes the length of the computed Parikh vector (the number of elements in the Parikh Vector).
- $|Model's Move|$: It denotes the length of the modeled trace in the final alignment (the number of elements in the modeled trace).
- *Alignment*: It denotes the final alignment obtained by ILPSDP technique.

10.4.2 ILPGA

This method is a genetic algorithm approach for alignment computation which the details is presented in Chapter 6.

10.4.2.1 Inputs

- This technique only accepts WF-nets. This approach of alignment computation first with the use of ILP finds a Parikh vector which is as similar as possible to the Parikh vector of observed trace like the previous method.
- Next, it generates a population of sequences (candidate solutions) based on the computed Parikh vector.

- It then starts to iterate over the population by applying cross-over and mutation operators that are defined in Chapter 6. The number of iterations can be specified by the user. The algorithm will be terminated whenever it reaches to the specified number of iterations.
- After initializing the mentioned parameters, you can click on Run to execute the approach. The following is a snapshot of ALI for the mentioned issues.

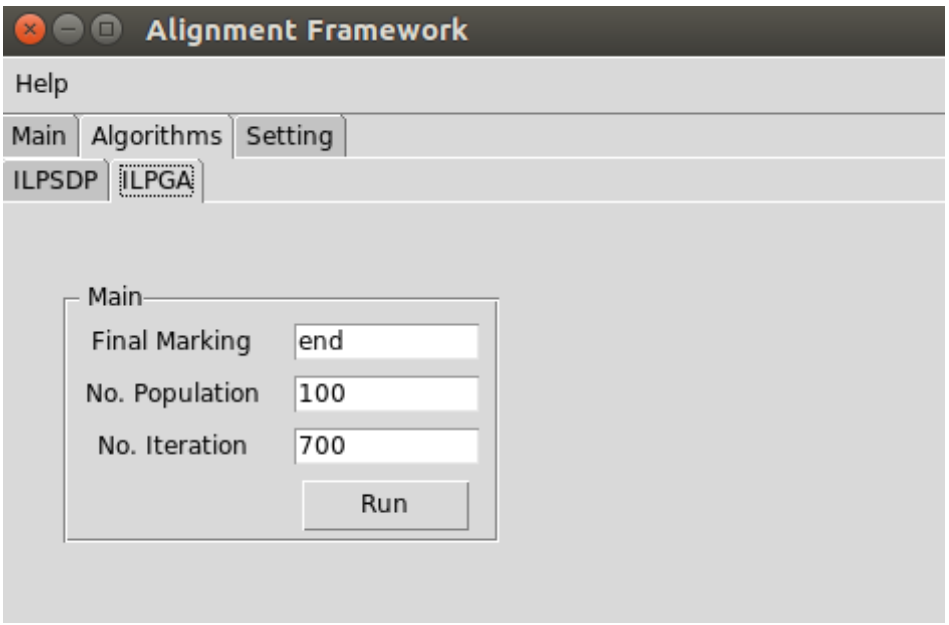


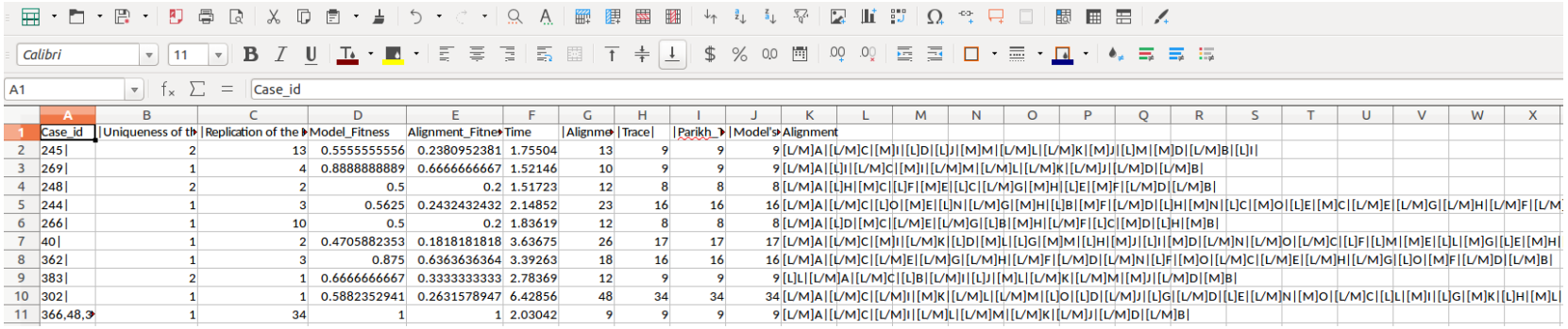
Figure 10.4: ILPGA settings

10.4.2.2 Outputs

ILPGA algorithm, provides two files. The first, is an Excel file with different information as shown in the following snapshot. Many of the features in the Excel file are described earlier for the previous algorithm, thus we don't mention them here. The descriptions of other columns are as follows:

- *Uniqueness of the Best Solution*: It shows for a given observed trace and specified settings, i.e., number of population and iterations, how many different solutions (best ones) with the same fitness values are obtained.
- *Replication of the Best Solution*: This column shows, for a given observed trace and specified settings, how many copies of the best solution(s) are available.

The second file is an XML file that contains all the best solutions for each observed trace. The following snapshot demonstrates it. In more



	A	B	C	D	E	F	G	H	I	J	K
1	Case_id	Uniqueness of th	Replication of the	Model_Fitness	Alignment_Fitne	Time	Alignme	Trace	Parikh_	Model's	Alignment
2	245	2	13	0.5555555556	0.2380952381	1.75504	13	9	9	9	[L/M]A|[L/M]C|[M]I|[L]D|[L]J|[M]M|[L/M]L|[L/M]K|[M]J|[L]M|[M]D|[L/M]B|[L]I|
3	269	1	4	0.8888888889	0.6666666667	1.52146	10	9	9	9	[L/M]A|[L]I|[L/M]C|[M]I|[L/M]M|[L/M]L|[L/M]K|[L/M]J|[L/M]D|[L/M]B|
4	248	2	2	0.5	0.2	1.51723	12	8	8	8	[L/M]A|[L]H|[M]C|[L]F|[M]E|[L]C|[L/M]G|[M]H|[L]E|[M]F|[L/M]D|[L/M]B|
5	244	1	3	0.5625	0.2432432432	2.14852	23	16	16	16	[L/M]A|[L/M]C|[L]O|[M]E|[L]N|[L/M]G|[M]H|[L]B|[M]F|[L/M]D|[L]H|[M]N|[L]C|[M]O|[L]E|[M]C|[L/M]E|[L/M]G|[L/M]H|[L/M]F|[L/M
6	266	1	10	0.5	0.2	1.83619	12	8	8	8	[L/M]A|[L]D|[M]C|[L/M]E|[L/M]G|[L]B|[M]H|[L/M]F|[L]C|[M]D|[L]H|[M]B|
7	40	1	2	0.4705882353	0.1818181818	3.63675	26	17	17	17	[L/M]A|[L/M]C|[M]I|[L/M]K|[L]D|[M]L|[L]G|[M]M|[L]H|[M]J|[L]I|[M]D|[L/M]N|[L/M]O|[L/M]C|[L]F|[L]M|[M]E|[L]L|[M]G|[L]E|[M]H|
8	362	1	3	0.875	0.6363636364	3.39263	18	16	16	16	[L/M]A|[L/M]C|[L/M]E|[L/M]G|[L/M]H|[L/M]F|[L/M]D|[L/M]N|[L]F|[M]O|[L/M]C|[L/M]E|[L/M]H|[L/M]G|[L]O|[M]F|[L/M]D|[L/M]B|
9	383	2	1	0.6666666667	0.3333333333	2.78369	12	9	9	9	[L]L|[L/M]A|[L/M]C|[L]B|[L/M]I|[L]J|[M]L|[L/M]K|[L/M]M|[M]J|[L/M]D|[M]B|
10	302	1	1	0.5882352941	0.2631578947	6.42856	48	34	34	34	[L/M]A|[L/M]C|[L/M]I|[M]K|[L/M]L|[L/M]M|[L]O|[L]D|[L/M]J|[L]G|[L/M]D|[L]E|[L/M]N|[M]O|[L/M]C|[L]L|[M]I|[L]G|[M]K|[L]H|[M]L|
11	366,48,3	1	34	1	1	2.03042	9	9	9	9	[L/M]A|[L/M]C|[L/M]I|[L/M]L|[L/M]M|[L/M]K|[L/M]J|[L/M]D|[L/M]B|

Figure 10.5: Output of ILPSDP algorithm in an Excel file

details, The XML file contains the information about the original event log, experiment setups (population size, number of iteration and optimizer name), and for each observed trace it contains many useful information as they are in the following picture.

Figure 10.6: XML output of ILPGA algorithm

10.4.3 Setting

- In this tab you can change the performance of ALI based on different settings. Since the presented algorithm and the future ones are based on ILP, therefore ALI has been integrated to work with both commercial and open-source LP solvers. The default value is set to "*PulP*", but using *Gurobi* is strongly recommended if you have installed it on your machine and want to use methods that are based on ILP heavily.

- Also it has been compatible with multi-core machines and takes the advantage of running on multi-CPU such that you can specify how many of CPUs are allowed to be used by ALI. It automatically detects the number of CPUs and except one of them, uses all the rest. The following picture shows the snapshot of mentioned issues.

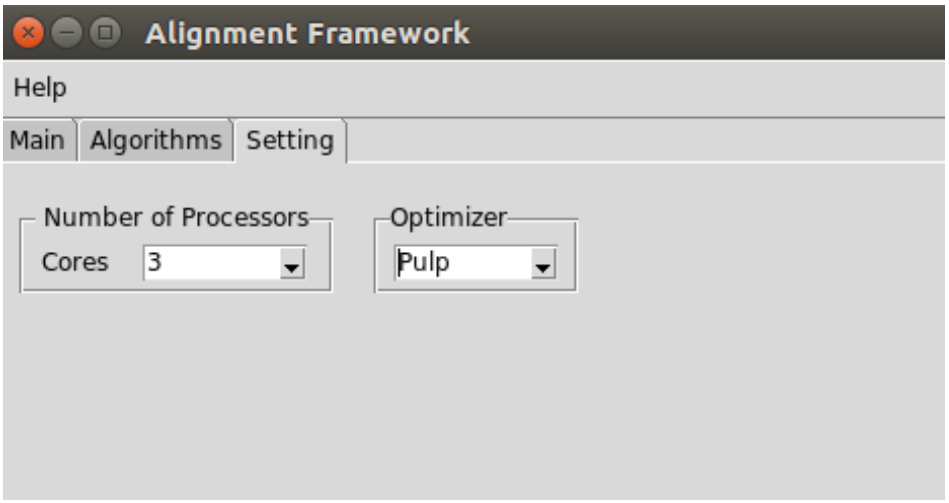


Figure 10.7: ALI settings

Appendices

Appendix A

Datasets

All the approaches presented in this thesis are examined through different datasets with various characteristics. Examples are artificial and realistic containing transitions with duplicate labels and from well-structured to completely Spaghetti. More specific they can be classified as follows:

- **Medium and large models**: These models are presented in Table A.1 [56]. These datasets contain models without or very rare deviations in event logs like $prAm6, prBm6$ and large models like $prDm6, prFm6$,
  $prEm6, prGm6$. These models do not have any loops, transitions with duplicate labels and any invisible transitions. The main challenge of these models is the size of models.

Table A.1: BPM2013 artificial benchmark datasets [56]

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$
prAm6	363	347	846	1200	No	31
prBm6	317	317	752	1200	Yes	43
prCm6	317	317	752	500	No	43
prDm6	529	429	1140	1200	No	248
prEm6	277	275	652	1200	No	98
prFm6	362	299	772	1200	No	240
prGm6	357	335	826	1200	No	143

- **Models with loops**: The specification of these models are presented in Table A.2. These models are synthetic and generated by PLG2 [17]. Different levels of noises have been injected to these models. All models contain nested loops, invisible transitions, but without duplicate names. The main challenge of these models are dealing with nested loops and very noisy event logs.

Table A.2: Artificial benchmark datasets containing loops

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$
M_1	40	39	92	500	No	13
M_2	34	34	80	500	No	17
M_3	108	123	276	500	No	37
M_4	36	52	106	500	No	26
M_5	35	33	78	500	No	34
M_6	69	72	168	500	No	53
M_7	65	62	148	500	No	37
M_8	17	15	36	500	No	17
M_9	47	55	120	500	No	44
M_{10}	150	146	354	500	No	58

- **Models contain transitions with duplicate names**: Details of these models are shown in Table A.3. These models consist of duplicate transition names with corresponding logs by different amount of errors, 25%, 35%, 50%, 75%, 25%, which were generated by PLG2 [17][1]. All models have nested loops, silent transitions and duplicate labels. The main challenge will be dealing with duplicate labels.

Table A.3: Models with duplicate transition names

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$	Duplicate Transitions
ML_1	27	35	74	500	No	28	2
ML_2	165	177	404	500	No	87	12
ML_3	45	45	106	500	No	26	2
ML_4	36	33	80	500	No	28	6
ML_5	159	172	390	500	No	42	14

- **Realistic models**: These models are realistic and some of them are generated from the corresponding real logs obtained from https://data.4tu.nl/repository/collection:event_logs_real. Benchmark Bank-transfer is taken from [95] and Documentflow benchmarks are taken from [102]. Some event logs from the last edition of the *BPI Challenge* were used, for which the models BPIC15_ 2, BPIC15_ 4, BPIC15_ 2 were generated using Inductive Miner plugin of ProM with noise threshold 0.99, 0.5 and 0.2, respectively. Details are shown in Table A.4. Some of these models are unstructured, i.e., Spaghetti. All

[1] At the time of generating models for the experiments, PLG2 in fact was unable to produce models containing duplicate labels from scratch, therefore the generated models and logs were modified in order to have transitions with duplicate labels.

models contain loop, invisible transitions but without duplicate labels. The main challenge is to handle unstructured feature of process models.

Table A.4: Realistic benchmark datasets

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$
Banktransfer	121	114	272	2000	No	58
Documentflow1	334	447	2059	12391	No	5
Documentflow2	337	456	2025	12391	No	5
BPIC15_ 2	78	420	848	832	No	53
BPIC15_ 4	178	464	954	1053	No	44
BPIC15_ 5	45	277	558	1156	No	51

Bibliography

[1] lpsolve : Open source (mixed-integer) linear programming system. 79

[2] *Integer Programming*, chapter 10, pages 588–631. Wiley-Blackwell, 2009. 40

[3] W. M. P. v. d. Aalst. *Process Mining: Data Science in Action*. Springer, second edition, 2016. 6, 7, 8, 9, 11, 24, 32

[4] W. Aalst, van der. Business process management: a comprehensive survey. *ISRN Software Engineering*, 2013, 2013. 8

[5] A. Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Technische Universiteit Eindhoven, 2014. 12, 13, 17, 19, 21, 34, 35, 69, 70, 79, 88, 111, 114, 116, 119, 120, 121, 141, 142, 144, 148, 149, 160, 162, 163, 168, 173, 178, 180, 182

[6] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst. Measuring precision of modeled behavior. *Inf. Syst. E-Business Management*, 13(1):37–67, 2015. 11

[7] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, EDOC '11, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society. 19

[8] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1996. 51

[9] S. Balaguer, T. Chatain, and S. Haar. Building occurrence nets from reveals relations. *Fundam. Inform.*, 123(3):245–272, 2013. 166

[10] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993. 50

[11] E. Best. Structure theory of petri nets: the free choice hiatus. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, pages 168–205, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. 29

[12] F. Bezerra, J. Wainer, and W. M. P. van der Aalst. Anomaly detection using process mining. In T. Halpin, J. Krogstie, S. Nurcan,

E. Proper, R. Schmidt, P. Soffer, and R. Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 149–161, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 10

[13] T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput.*, 4(4):361–394, Dec. 1996. 49

[14] R. P. J. C. Bose and W. M. P. van der Aalst. Trace alignment in process mining: Opportunities for process diagnostics. In *Proceedings of the 8th International Conference on Business Process Management*, BPM'10, pages 227–242, Berlin, Heidelberg, 2010. Springer-Verlag. 22

[15] J. C. A. M. Buijs. *Flexible Evolutionary Algorithms for Mining Structured Process Models*. PhD thesis, Technische Universiteit Eindhoven, 2014. 19

[16] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*, pages 1–8, 2012. 123

[17] A. Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.*, pages 1–6, 2016. 205, 206

[18] A. Burattin and J. Carmona. A framework for online conformance checking. In E. Teniente and M. Weidlich, editors, *Business Process Management Workshops*, pages 165–177, Cham, 2018. Springer International Publishing. 21

[19] J. M. Colom, E. Teruel, M. Silva, and S. Haddad. *Structural Methods*, pages 277–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. 27

[20] M. de Leoni and A. Marrella. Aligning real process executions and prescriptive process models through automated planning. *Expert Syst. Appl.*, 82:162–183, 2017. 12, 141

[21] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, Apr 2007. 9

[22] J. Desel and J. Esparza. Reachability in cyclic extended free-choice systems. *TCS 114, Elsevier Science Publishers B.V.*, 1993. 31

[23] J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, Cambridge, Great Britain, 1995. 29, 87, 97

[24] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers. *Introduction to Business Process Management*, pages 1–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. 8

[25] P. H. C. Eilers and B. D. Marx. Flexible smoothing with b-splines and penalties. *STATISTICAL SCIENCE*, 11:89–121, 1996. 135

[26] J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, (16):159–189, 2000. 31

[27] D. Fahland and W. M. P. van der Aalst. Repairing process models to reflect reality. In A. Barros, A. Gal, and E. Kindler, editors, *Business Process Management*, pages 229–245, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 10

[28] D. Fahland and W. M. P. van der Aalst. Model repair - aligning process models to reality. *Inf. Syst.*, 47:220–243, 2015. 10, 14, 57

[29] S. Farnham, S. U. Kelly, W. Portnoy, and J. L. K. Schwartz. Wallop: designing social software for co-located social networks. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10 pp.–, Jan 2004. 11

[30] L. Garca-Bauelos, N. R. T. P. van Beest, M. Dumas, M. L. Rosa, and W. Mertens. Complete and interpretable conformance checking of business processes. *IEEE Transactions on Software Engineering*, 44(3):262–290, March 2018. 22

[31] M. T. Gómez-López, D. Borrego, J. Carmona, and R. M. Gasca. Computing alignments with constraint programming: The acyclic case. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016 Satellite event of the conferences: 37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and 16th International Conference on Application of Concurrency to System Design ACSD 2016, Torun, Poland, June 20-21, 2016.*, pages 96–110, 2016. 19

[32] I. Gurobi Optimization. Gurobi optimizer reference manual, 2016. 79, 116

[33] C. W. Gunther, A. Rozinat, and K. V. Uden. Monitoring deployed application usage with process mining, 2014. 10

[34] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. 44

[35] J. H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984. 40

[36] B. J. Hommes and V. van Reijswoud. Assessing the quality of business process modelling techniques. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.1–, Jan 2000. 24

[37] S. hyuk Cha. A genetic algorithm for constructing compact binary decision trees, 2009. 48

[38] IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data.* McGraw-Hill Osborne Media, 1st edition, 2011. 6

[39] R. Johnson, D. Pearson, and K. Pingali. The program structure tree: Computing control regions in linear time. *SIGPLAN Not.*, 29(6):171–185, June 1994. 166

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983. 40

[41] M. Koorneef, A. Solti, H. Leopold, and H. A. Reijers. Automatic root cause identification using most probable alignments. In *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, pages 204–215, 2017. 123

[42] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Scalable process discovery with guarantees. In K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, editors, *Enterprise, Business-Process and Information Systems Modeling: 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Proceedings (Lecture Notes in Business Information Processing, Volume 214)*, pages 85–101, Cham, Switzerland, 2015. Springer. 9

[43] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Scalable process discovery and conformance checking. *Software & Systems Modeling*, 17(2):599–631, May 2018. 12, 22

[44] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets.* Cambridge University Press, New York, NY, USA, 2nd edition, 2014. 9

[45] X. Lu, R. Mans, D. Fahland, and W. M. P. van der Aalst. Conformance checking in healthcare based on partially ordered event data. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation, ETFA 2014, Barcelona, Spain, September 16-19, 2014*, pages 1–8, 2014. 14, 57

[46] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016. 124

[47] R. Mans, H. Schonenberg, M. Song, W. M. P. Aalst, and P. Bakker. Application of process mining in healthcare a case study in a dutch hospital, 01 2008. 10

[48] R. S. Mans, W. M. P. van der Aalst, and R. J. B. Vanwersch. *Applications of Process Mining*, pages 53–78. Springer International Publishing, Cham, 2015. 11

[49] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competi-

tion, and productivity. Technical report, McKinsey Global Institute, June 2011. 7

[50] J. Mendling, G. Neumann, and W. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 113–130, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 12

[51] Z. Michalewicz. *How to Solve It: Modern Heuristics 2e.* Springer-Verlag, Berlin, Heidelberg, 2010. 13, 44

[52] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. 46

[53] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, pages 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. 48

[54] J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst. Conformance checking in the large: Partitioning and topology. In F. Daniel, J. Wang, and B. Weber, editors, *Business Process Management*, pages 130–145, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 20

[55] J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst. Hierarchical conformance checking of process models based on event logs. In J.-M. Colom and J. Desel, editors, *Application and Theory of Petri Nets and Concurrency*, pages 291–310, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 20

[56] J. Munoz-Gama, J. Carmona, and W. M. P. Van Der Aalst. Single-entry single-exit decomposed conformance checking. *Inf. Syst.*, 46:102–122, Dec. 2014. 20, 57, 117, 184, 185, 205

[57] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, Apr. 1989. 25, 26, 28, 31, 72

[58] R. Neapolitan. *Foundations Of Algorithms*, pages 138–146. Jones and Bartlett Publishers, Inc., USA, 5th edition, 2014. 104, 140

[59] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970. 87, 104, 105, 140, 166, 169, 173

[60] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981. 27

[61] A. Piszcz and T. Soule. Genetic programming: Optimal population sizes for varying complexity problems. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 953–954, New York, NY, USA, 2006. ACM. 126

[62] A. Polyvyanyy, S. Smirnov, and M. Weske. *The Triconnected Abstraction of Process Models*, pages 229–244. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 20, 165

[63] A. Polyvyanyy, J. Vanhatalo, and H. Völzer. Simplified computation and generalization of the refined process structure tree. In *7th International Conference on Web Services and Formal Methods*, WS-FM'10, pages 25–41, Berlin, Heidelberg, 2011. 20, 166

[64] J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370, Jun 1996. 48

[65] D. Reißner, R. Conforti, M. Dumas, M. La Rosa, and A. Armas-Cervantes. Scalable conformance checking of business processes. In H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, and R. Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 607–627, Cham, 2017. Springer International Publishing. 21

[66] D. Reißner, R. Conforti, M. Dumas, M. L. Rosa, and A. Armas-Cervantes. Scalable conformance checking of business processes. In *OTM CoopIS, , Rhodes, Greece*, pages 607–627, 2017. 12, 141, 142, 144, 145, 148, 149

[67] D. Reißner, R. Conforti, M. Dumas, M. L. Rosa, and A. Armas-Cervantes. Scalable conformance checking of business processes. Paper submitted to "International Conference on Business Process Management (BMP 2017)" in Barcelona, Spain., March 2017. 88, 116, 119, 120

[68] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008. 18, 88

[69] D. Ruppert, M. P. Wand, and R. J. Carroll. *Scatterplot Smoothing*, page 5790. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2003. 135

[70] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993. 51

[71] M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In Reisig, W. and Rozenberg, G., editors, *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, volume 1491, pages 309–373. Springer-Verlag, 1998. 30, 31, 41, 66

[72] R. E. Tarjan and J. Valdes. Prime subprogram parsing of a program. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, pages 95–105, New York, NY, USA, 1980. ACM. 20

[73] F. Taymouri. ALI: Alignment for Large Instances, 2017. x, 55, 85, 88, 116, 141, 153

[74] F. Taymouri and J. Carmona. Model and event log reductions to boost the computation of alignments. In *Data-Driven Process Discovery and Analysis - 6th IFIP WG 2.6 International Symposium, SIMPDA 2016, Graz, Austria, December 15-16, 2016, Revised Selected Papers*, pages 1–21, 2016. 15, 55, 79, 153

[75] F. Taymouri and J. 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. 15

[76] F. Taymouri and J. Carmona. Computing alignments of well-formed process models using local search. *Submitted to ACM Transactions on Software Engineering and Methodology*, 2018. 15, 85

[77] F. Taymouri and J. Carmona. An evolutionary technique to approximate multiple optimal alignments. In *16th International Conference of Business Process Management (BPM), Sydney, Australia. September 9-14, Brazil, September 18 - 22*, 2018. 15, 85

[78] W. M. P. v. Aalst, K. M. v. Hee, J. M. v. Werf, and M. Verdonk. Auditing 2.0: Using process mining to support tomorrow's auditor. *Computer*, 43(3):90–93, March 2010. 9

[79] W. van der Aalst, H. Reijers, A. Weijters, B. van Dongen, A. A. de Medeiros, M. Song, and H. Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713 – 732, 2007. 11

[80] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, Jul 2003. 24

[81] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing: applicable formal methods*, 23(3):333–363, 2010. 28

[82] W. van der Aalst, A. Weijter, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:2004, 2003. 9

[83] W. M. van der Aalst, A. Andriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. Gunther, A. Guzzo, P. Harmon, A. H. ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. L. Rosa, F. Maggi, D. Malerba, R. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. M. Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pointieri, J. Ribeiro, A. Rozinat, H. S. Perez, R. S. Perez,

M. Sepulveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. T. Wynn. Process mining manifesto. In *7th International Workshop on Business Process Intelligence (BPI 2011)*, pages 169–194, Campus des Cézeaux, Clermont-Ferrand, 2012. Springer-Verlag. 7

[84] W. M. P. van der Aalst. Making work flow: On the application of petri nets to business process management. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002*, pages 1–22, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. 28

[85] W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. 34

[86] W. M. P. van der Aalst. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013. 57

[87] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. Genetic process mining. In *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer, 2005. 123

[88] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.*, 23(3):333–363, 2011. 87

[89] W. M. P. van der Aalst and H. M. W. Verbeek. Process discovery and conformance checking using passages. *Fundam. Inf.*, 131(1):103–138, Jan. 2014. 9

[90] B. van Dongen, J. Carmona, Th. Chatain, and F. Taymouri. Aligning modeled and observed behavior: A compromise between complexity and quality. In E. Dubois and K. Pohl, editors, *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE'17)*, volume 10253 of *Lecture Notes in Computer Science*, Essen, Germany, June 2017. Springer. 15, 55

[91] B. F. van Dongen. Efficiently computing alignments. In M. Weske, M. Montali, I. Weber, and J. vom Brocke, editors, *Business Process Management*, pages 197–214, Cham, 2018. Springer International Publishing. 82

[92] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The prom framework: A new era in process mining tool support. In *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets*, ICATPN'05, pages 444–454, Berlin, Heidelberg, 2005. Springer-Verlag. 19, 55, 69, 82

[93] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, Oct 2017. 21

[94] S. K. L. M. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen. Event-based real-time decomposed conformance analysis. In R. Meersman, H. Panetto, T. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T. Sellis, editors, *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, pages 345–363, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. 21

[95] S. K. L. M. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen. Event-based real-time decomposed conformance analysis. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, pages 345–363, 2014. 206

[96] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, and B. Baesens. Determining process model precision and generalization with weighted artificial negative events. *IEEE Trans. Knowl. Data Eng.*, 26(8):1877–1889, 2014. 88

[97] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag. 20

[98] B. Vázquez-Barreiros, M. Mucientes, and M. Lama. Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Inf. Sci.*, 294:315–333, 2015. 123

[99] H. M. W. Verbeek and W. M. P. van der Aalst. *Merging Alignments for Decomposed Replay*, pages 219–239. Springer International Publishing, Cham, 2016. 20

[100] C. R. Vogel. *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. 135

[101] I. Weber, A. Rogge-Solti, C. Li, and J. Mendling. CCaaS: Online conformance checking as a service. In *International Conference on Business Process Management, Demo Track*, pages 45–49, Innsbruck, Austria, Aug. 2015. 22

[102] J. D. Weerdt, S. K. L. M. vanden Broucke, J. Vanthienen, and B. Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25(12):2708–2720, 2013. 206

[103] M. Weidlich, A. Polyvyanyy, J. Mendling, and M. Weske. Causal behavioural profiles - efficient computation, applications, and evaluation. *Fundam. Inf.*, 113(3-4):399–435, Aug. 2011. 21, 165, 166

[104] S. J. Zelst, B. F. Dongen, W. M. Aalst, and H. M. Verbeek. Discovering workflow nets using integer linear programming. *Computing*, 100(5):529–556, May 2018. 9