RWTH AACHEN UNIVERSITY

This work was submitted to:

Chair of Process and Data Science (PADS - Informatik 9), RWTH Aachen University

---

Efficient State-Space Traversal in Alignment Computation

Master's Thesis

Author: **Tian Li**

Student ID: **391589**

Supervisor: Dr. Sebastiaan J. van Zelst

Examiners: Prof. Wil van der Aalst
Prof. Stefan Decker

Registration Date: 2021-10-13

Submission Date: 2022-05-04

Abstract

With the advanced support of information systems, processes become increasingly digitized in companies. As event data describing the digital footprint of these processes is available in the underlying databases, these data can be analyzed to reveal what really happened during the execution of the process. However, real-world processes often deviate from the desired specification and can lead to misleading judgment, thus conformance checking evaluates whether event logs and process models conform to each other, and provides insight into the process. Alignments have proven to be a very useful technique for calculating conformance metrics, particularly, diagnostics. In this thesis, we explore several variations of alignment computation techniques, which either approximate or compute optimal alignments. Compared to traditional techniques, our proposed approaches aim at improving search efficiency mainly by reducing the number of states and arcs traversed during the search. This is achieved by introducing a caching set for the search, and propagating sequence information for every state expanded during the search. We evaluated the techniques with publicly available data sets and compared the results with existing approaches. The results show that the proposed algorithm computes the (optimal) alignment faster by preventing redundant exploration of the state space.

Contents

Chapter 1

Introduction

Process Mining is a research field that focuses on extracting valuable knowledge from event logs readily available in today's enterprise systems, Event logs can be used to conduct three types of analyses, namely, *Process Discovery*, *Conformance Checking*, and *Process Enhancement* [1]. *Process Discovery* takes an event log and generates a process model without using any apriori information [2]. *Conformance Checking* refers to the analysis of the relation between the intended behaviour of a process as described in a process model and event logs that have been recorded during the execution of the process [3]. *Process Enhancement* aims to change or improve the apriori model [2].

In this thesis, we concentrate solely on conformance checking, which measures the deviations by comparing process executions with their normative behavior. For instance, an event log can be "replayed" on a given process model to uncover potential deception or bottleneck in a company, and exposing deviations between the model and process may facilitate better working instructions. Also, business process managers benefit by determining whether the process is executed as planned or abides its prescribed behavior.

Alignments are one of the most fundamental conformance checking artifacts, which captures how the event log can be replayed on the process model. It may be the case that some behaviors in the log is not described by the model and vice versa, thus alignments can describe the mismatches between a sequence of activities in an event log and an execution sequence from a model.

In the remainder of this chapter, we first discuss the motivation for improving alignment computation techniques in Section 1.1. On top of the problem statement in Section 1.2 we elaborate the research questions in Section 1.3, which lay the foundation for our research goals in Section 1.4. We subsequently highlight the main contributions in Section 1.5 and present the thesis structure in Section 1.6.

1.1 Motivation

In this section, we elaborate on the necessity of improving the state-of-the-art alignment computation technique. Recent advancements in alignment computation have led to the *split-point-based alignments algorithm* [4] that extends traditional A^*-based search with so-called split points. Using these trace splits, the algorithm splits the marking equation

into several smaller sub-problems. Although this technique greatly reduces the number of linear programming problems solved during the search, it abandons all information obtained during the previous round of the search.

Therefore, we argue that this split-based technique has much room for improvement, i.e., by reusing previously obtained results, it potentially process fewer markings and arcs during the search. The reduction of repeated exploration leads to more efficient state space traversal as shown in Table 1.1. Model 1 to 4 refer to process models mined with Inductive Miner (infrequent) from ProM [5], with noise thresholds at 0.05, 0.20, 0.40 and 0.80.

Table 1.1: Time reduction with proposed approach

	Models mined with Inductive Miner (infrequent)			
Event log	Model 1	Model 2	Model 3	Model 4
Hospital bill [6]	26%	19%	15%	19%
Sepsis [7]	30%	15%	14%	24%
CCC2019 [8]	73%	77%	71%	48%
CCC2020-A [9]	38%	15%	22%	10%
...				

1.2 Problem Statement

The current split-point-based algorithm is not satisfying as it searched the underlying state space in an inefficient manner. When computing heuristic, it uses an extended marking equation to guarantee certain transitions are enabled before firing, i.e., it ensures that at input places of the transition exist enough tokens. Although the heuristic computed is more accurate and guides the search better, the frequent restart procedures lead to repeated exploration of markings and arcs in the state space, which undermines the search efficiency.

1.3 Research Questions

The analysis of how previously obtained results can be reused on top of the split-point based algorithm leads to the following research questions:

RQ1. How to efficiently store infeasible states without a feasible solution vector during the search?

RQ2. How to measure the feasibility of previous infeasible markings with a new solution vector?

RQ3. How to exploit previously obtained results to avoid redundant restarts?

RQ4. Does the strategy of reusing previously obtained results contribute to the enhanced performance of the split-point-based algorithm?

RQ5. Does a trade-off exist between always restarting from scratch and never restarting?

RQ6. Are the optimal alignments guaranteed with the proposed reusing strategy?

1.4 Research Goals

In the context of the aforementioned questions, we define the research goals as follows:

RG1. Reuse previously obtained information with a novel caching strategy.

RG2. Measure the performance of the split-point-based algorithm before and after applying the caching strategy.

RG3. Investigate the trade-off between restarting and not restarting by analyzing the performance of the reusing the caching strategy.

1.5 Contributions

This thesis has contributed mainly in the following three aspects:

C1. An incremental approach on top of a current alignment computation technique.

C2. Investigate and prove the effectiveness of the proposed approach with real-life datasets.

1.6 Thesis Structure

In this chapter, we present the overview of conformance checking and introduce the research goals and contributions of the thesis. The remainder of the thesis is structured as follows: In Chapter 2, we discuss related work in conformance checking. In Chapter 3, we introduce preliminary information necessary notations and definitions, as well as the alignment computation techniques. In Chapter 4, we elaborate on the implementation of the methods. In Chapter 5, we evaluate the approach with various data sets and compare the results with existing techniques. In In Chapter 6, we discuss the observations of our general approach. Finally, we conclude our work and suggest possible future work in Chapter 7.

Chapter 2

Related Work

In this thesis, we concentrate mainly on work related to conformance checking. In Section 2.1 we briefly discuss token replay techniques, then in Section 2.2 we review alignments-based techniques.

2.1 Token Replay Algorithms

Token replay is one of the early algorithms, which laid the foundation for conformance checking [10]. The approach replays traces in the event log on a given process model. During replay, missing tokens will be added if the replay gets stuck, which enables the firing of a transition to advance the current state to the next state. Finally, the remaining tokens in the model will be counted. Fitness is calculated based on missing and remaining tokens, together with generated and consumed tokens.

Token-based replay technique is simple and easy to implement, but there are some drawbacks. If the trace does not fit, the approach does not create a path in the model to map observed behavior and modeled behavior. Also, frequent deviations lead to token-flooding, which allows for undesired arbitrary behavior. Although the authors in [11] propose to scale the method with an additional identification procedure for superfluous tokens, it does not guarantee fitness results. In the following section, we introduce alignment-based techniques, which provide a complete path for every trace in the process model and explicitly show where deviations take place.

2.2 Alignment-based Algorithms

In this section, we discuss alignment-based techniques, which is the de-facto standard in conformance checking [12]. The framework of trace alignments is first introduced [13], which discusses various scenarios of misalignments. The technique not only maps observed behavior in the trace to modeled behavior in the model but also finds a path in the model. We classify alignments techniques into two categories: computing optimal alignments and approximating optimal alignment. Given an event log and a process model, the former algorithm type computes optimal alignments while the latter approximates optimal alignments.

2.2.1 Alignments Computation Algorithms

In [14], the author proposes computing optimal alignments as constructing a synchronous product net from a given Petri net and a trace from an event log, which transforms the conformance checking problem into solving the shortest path problem of state space by using A^* search algorithm. In addition, the A^* search utilizes the marking equation as the underestimation function.

To improve the scalability of this method, the authors in [15, 16] propose different tuning and parametrization techniques. The results show that restrictions on the model move during search promotes the search efficiency.

Another approach in [4] utilizes the original trace to guarantee the progress of the search. When the search gets stuck, the algorithm finds the maximum event index that has been explored and uses it as a split point. Then the marking equation is split into smaller sub-problems and the technique seeks to provide a better underestimation function for the initial marking. Later in [17] the authors propose to construct structural or behavioral rules from the process model, so that the split points are first ascertained and used to split the trace before alignments computation.

While the aforementioned algorithms are designed for offline settings that handle static historical data, online process mining techniques is introduced in [18] to analyze in-progress cases of event streams. In [19], the authors propose to compute prefix alignments every time a new event is observed, which approximated the result for incomplete cases. Since the approach did not guarantee optimality, it is improved in [20] by utilizing specific properties of the search space regarding prefix-alignments computation, which continues the calculation of prefix-alignments on an event stream from previous results. Moreover, in [21] three techniques are proposed to deal with memory scarcity and avoid the missing-prefix problem for prefix alignments.

Moreover, the previously mentioned techniques focus purely on control flow in a process model, some researchers have advanced their approaches from other perspectives, i.e., taking context data into consideration. In [22], the authors construct an additional ILP problem for each log trace, so that alignments are extended to incorporate other dimensions like date time and resources. Another approach in [23] allows for arbitrary cost functions covering all perspectives, such as data dependencies, resource assignments and time constraints, which balances different perspectives with customizable settings.

However, the calculation of alignments is computationally complex since the shortest path problem on the reachability graph of the synchronous product net is solved, and the complexity grows non-linearly with the size of the graph [4].

2.2.2 Alignments Approximation Algorithms

To handle large and complex conformance checking problems, a number of approaches applied decomposition techniques. Based on the divide and conquer strategy, the method proposed in [24] applies a recursive scheme to solve multiple integer linear programming problems. It identifies deviations between sets of transitions and computes partial alignment, which handles large conformance instances in an efficient manner. Another alignments technique using decomposition is introduced in [25], which deals with large conformance instances. The Petri net is decomposed into smaller sub-nets to align with certain

parts of the trace. Although it does not provide an alignment for a full trace, it identifies deviations in the specific part of the process model faster.

In addition to decomposition, some works resolve alignments based on trace sampling. The authors in [26] present a statistical approach to trace sampling and conformance approximation, which guarantees the accuracy of the estimated conformance result. Similarly, the algorithm in [27] provides approximated alignments values by considering a subset of the process model behavior instead of all behavior. The subset is determined by simulating the process model or computing optimal alignments for some candidate traces in the model.

Furthermore, various works introduced additional constraints for alignments computation. In [28], a method is proposed to preprocess the model and then apply relaxation labeling on the partial order representation of a process model. The approach proposed in [29] introduces unskippable actions and a max synchronization cost function to guide the state space traversal. It maximizes the number of synchronous events rather than minimizing discrepancies between the trace and the model.

Although these techniques are less time-consuming, the conformance checking results are not guaranteed to be optimal.

Chapter 3

Preliminaries

In this chapter, we discuss multiple definitions and notations that are used in the remainder of the thesis. We introduce general definitions in Section 3.1, then we define the notion of *Event Logs* in Section 3.2. *Petri nets* are presented in Section 3.3. In Section 3.4, we define *Alignments* and subsequently in Section 3.5, we illustrate a basic alignment computation technique. Finally in Section 3.6, we introduce the *split-point-based algorithm* to lay the foundation of our approach.

3.1 General

The set of all possible multi-sets over set X is denoted as $\mathcal{B}(X)$ and the set of all possible sequences over set X is X^*. We denote the empty sequence as $\langle\rangle$, and the concatenation of sequences σ_1 and σ_2 as $\sigma_1{\cdot}\sigma_2$.

Given tuple $\overline{x}{=}(x_1,\ldots,x_n)$ of Cartesian product $X_1{\times}X_2{\times}\ldots{\times}X_n$, let $\pi_i(\overline{x}){=}x_i{\in}X_i$ for $1{\leq}i{\leq}n$. Given a tuple $\bar{t}{\in}X{\times}Y{\times}Z$, we implicitly define the index of the projection to derive $\pi_1(\bar{t}){\in}X$, $\pi_2(\bar{t}){\in}Y$, $\pi_3(\bar{t}){\in}Z$. We extend the notation to sequences, thus for a given sequence $\overline{\sigma}{\in}(X_1{\times}\ldots{\times}X_n)^*$ of length k where $\overline{\sigma}{=}\langle(x_1^1,x_2^1,\ldots,x_n^1),(x_1^2,x_2^2,\ldots,x_n^2),\ldots,(x_1^k,x_2^k,\ldots,x_n^k)\rangle$, we have $\pi_i(\overline{\sigma}){=}\langle x_i^1,x_i^2,\ldots,x_i^k\rangle\ {\in}X_i^*$ for $\forall i{\in}\{1,2,\ldots,n\}$. Let $\sigma{=}\langle x_1,\ldots,x_k\rangle{\in}X^*$ be a sequence and f be a function with $f:X{\to}Y$, we define $\pi^f:X^*{\to}Y^*$ with $\pi^f(\sigma){=}\langle f(x_1),\ldots,f(x_k)\rangle$. Given $Y{\subseteq}X$ we define $\downarrow_Y{:}\,X^*{\to}Y^*$ recursively with $\downarrow_Y(\langle\rangle){=}\langle\rangle$ and $\downarrow_Y(\langle x\rangle\cdot\sigma){=}\langle x\rangle{\cdot}\downarrow_Y(\sigma)$ if $x{\in}Y$ and $\downarrow_Y(\langle x\rangle\cdot\sigma){=}\downarrow_Y(\sigma)$ if $x{\notin}Y$. $\downarrow_Y(\sigma)$ is written as $\sigma_{\downarrow Y}$.

Given a matrix C with m rows and n columns, $C_{i,j}$ denotes the element on i-th row and j-th column$(1{\leq}i{\leq}m,1{\leq}j{\leq}n)$. C^{T} denotes the transpose of C.

Let $\Sigma{=}\{a_1,a_2,\ldots,a_k\}$ be an alphabet. The Parikh vector of a word x is defined as the function $\psi:\Sigma^*{\to}\mathbb{N}^k$, defined by $\psi(x){=}(\#a_1(x),\#a_2(x),\ldots,\#a_k(x))$. Thus $\psi(x)$ denotes the number of occurrences of each symbol in word x [30].

3.2 Event Logs

In the IT systems of a company, the execution of business processes is stored. Such data is easily extracted and describes instances of the process, i.e., the sequence of activities

that have been performed over time. We refer to such a collection of activity sequences as an *event log*, and the sequence of executed process activities for a process instance as a *trace*.

We present a fragment of an example event log in Table 3.1, which represents a simplified recording of handling item repair or replacement process in a company. We assume that events are related to an *activity*, such as activities *receive return*, *repair* and *send repair* in Table 3.1. Each *case* represents a *process instance*, which consists of a sequence of events. For example, case id 485 describes the case related to a repair process with the following order of activities: first Jack *received return*, then it got *replaced* and *registered* by Ben, and Amy *informed progress to the customer*, finally John *send repair*. Additionally, *timestamps* are recorded for events in the log.

Since the problem we deal with only considers the sequential scheduling of activities recorded in the context of process instances, we simplify the definition of the event log and formalize it as follows:

Definition 3.1 (**Event Log**)**.**
Let $\mathcal{A}$ denotes the set of activities, we define an event log L as a multi-set of traces over $\mathcal{A}$, i.e., $L \in \mathcal{B}(\mathcal{A}^*)$.

Table 3.1: A fragment of example event log

Event-id	**Case-id**	**Activity**	**Resource**	**Time-stamp**
...	...	...	...	...
1	484	receive return(a)	Jack	09/10/2021 10:27
1	485	receive return(a)	Jack	10/10/2021 10:13
3	485	get replacement(b)	Ben	11/10/2021 11:21
4	485	register return(d)	Ben	11/10/2021 11:21
2	485	inform progress to customers(e)	Amy	12/10/2021 10:51
3	484	inform progress to customers(e)	Ben	12/10/2021 11:14
6	485	send replacement(h)	John	12/10/2021 15:01
6	484	send repair(g)	John	12/10/2021 15:22
...	...	...	...	...

3.3 Petri nets

Petri nets are a commonly investigated process modeling language to describe processes, which are similar to UML diagrams and Business Process Model and Notation (BPMN) used in industry.Petri nets are not only simple and intuitive but also define exactly the mathematical theory and execution semantics of process, including choice, iteration, and concurrency.

A Petri net is a bipartite graph made up of *places*, *transitions*, which are depicted as circles and rectangles. We present an example Petri net N_1 in Fig. 3.1, which describes a simplified repair/replacement process in a company. As shown in N_1, after place p_1 we reach transition t_1 with label a. Then with an *AND-split*, p_2 and p_3 are connected with t_1. The *XOR-split* after p_2 leads to transitions t_2 and t_3 with label b and c. For both transitions t_5 and t_6 with label e they require *AND-join*. In addition, transition t_7 leads to a loop in the Petri net. We define *Labeled Petri nets* as follows:

Definition 3.2 (**Labeled Petri nets**).
A labeled Petri net $N=(P,T,F,A,\lambda)$ *where* P *denotes a finite set of places,* T *denotes a finite set of transitions such that* $P\cap T=\emptyset$*, and* $F\subseteq(P\times T)\cup(T\times P)$ *is a set of directed arcs called the flow relation.* A *is a set of activity labels and* λ *is a labeling function, such that* $\lambda\in T\to A$ *with* $\tau\notin A$.

According to Definition 3.2, Petri net N_1 is formalized as followed:

1. $P=\{p_1,p_2,p_3,p_4,p_5,p_6,p_7\}$.
2. $T=\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9\}$.
3. $F=\{(p_1,t_1),(t_1,p_2),(t_1,p_3),(p_2,t_2),(p_2,t_3),(p_3,t_4),(p_3,t_5),(t_2,p_4),(t_3,p_4),(p_4,t_5),(p_4,t_6),(t_4,p_5),(p_5,t_6),(t_6,p_6),(p_6,t_8),(p_6,t_9),(p_6,t_7),(t_8,p_7),(t_9,p_7),(t_7,p_3)\}$.
4. $A=\{a,b,c,d,e,f,g,h\}$.
5. The label function for N_1 is $\lambda=\{\lambda(t_1)=a,\lambda(t_2)=b,\lambda(t_3)=c,\lambda(t_4)=d,\lambda(t_5)=e,\lambda(t_6)=e,\lambda(t_7)=\tau,\lambda(t_8)=g,\lambda(t_9)=h\}$.

As shown in N_1, different transitions may have the same label, i.e., both transitions t_5 and t_6 share the same label e. Sometimes particular transitions are not observable, which adopt the label τ. Thus we refer to transition t_7 as unobservable transition, which is also referred to as τ or invisible transition.

In addition, the state of a Petri net is determined by the distribution of tokens over places, which is referred to as marking [2]. Each place can hold arbitrary number of tokens, which are depicted as black circles. The marking in N_1 is $[p_1]$, i.e., a multi-set containing only one token. We define marked Petri net as follows:

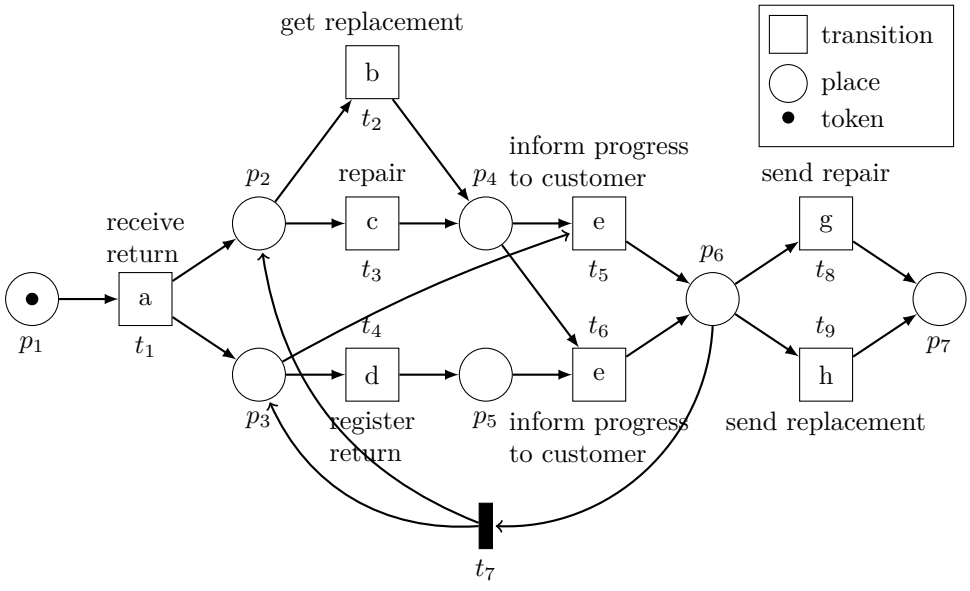


Fig. 3.1: Petri net N_1: a simplified product repair/replace process

Definition 3.3 (**Marked Labeled Petri net**).
A marked labeled Petri net $N=(P,T,F,A,\lambda,M)$ *where* (P,T,F,A,λ) *is the Petri net defined in Definition 3.2.* $M\in\mathcal{B}(P)$ *is a multi-set over* P*, which denotes the marking of the*

Petri net.

The behavior of a marked Petri net is defined by the *firing rule*, which specifies that a transition is *enabled* if all of its input places contains a token. An enabled transition can *fire* to consume a token from each input place and produce a token for each output place. For example, in N_1 transition t_1 is enabled at marking $[p_1]$. If we fire t_1, the token in p_1 will be consumed and two tokens will be produced for place p_2 and p_3. Thus firing t_1 results in marking $[p_2, p_3]$, from which transition t_1 is no longer enabled whereas transitions t_2, t_3, t_4 are enabled. We formalize the firing rule as follows:

Definition 3.4 (**Firing rule and Firing Sequence [2]**).
Let $N{=}(P, T, F, A, \lambda, M)$ be a marked labeled Petri net. The set of transitions T and places P denotes nodes. A node a is called an input node of another node b if and only if $(a, b){\in}F$, and b is an output node of a if and only if $(a, b){\in}F$. For any $a{\in}P{\cup}T, {\bullet}a{=}\{b|(b, a){\in}F\}$, which denotes the set of all input nodes of a. $a{\bullet}{=}\{b|(a, b){\in}F\}$, which denotes the set of all output nodes of a. Transition $t{\in}T$ is enabled, if and only if ${\bullet}t{\leq}M$. $(N, M)[t\rangle(N, M')$ denotes that transition t is enabled at marking M and firing t results in marking M'. Let M_0 be the initial marking, a sequence is called a firing sequence of N if and only if, there exist markings $M_1, \ldots, M_k$ and transitions $t_1, \ldots, t_k{\in}T$ such that $\sigma{=}\langle t_1, \ldots, t_k\rangle$ and $0{\leq}i{<}n, (N, M_i)[t_{i+1}\rangle(N, M_{i+1})$.

For instance, in N_1, ${\bullet}p_7{=}\{t_8, t_9\}$, $p_2{\bullet}{=}\{t_2, t_3\}$. Let the initial marking $m_i{=}[p_1]$ and the final marking $m_f = [p_7]$, there is a firing sequence $\sigma{=}\langle t_1, t_2, t_4, t_6, t_8\rangle$ such that $m_i[\sigma\rangle m_f$. For a Petri net, we refer to the set of all reachable markings from the initial marking as its state space.

The incidence matrix is a common graph representation in graph theory, we define the incidence matrix of a Petri net as follows:

Definition 3.5 (**Incidence Matrix**).
A Petri net $N{=}(P, T, F)$ is denoted as a incidence matrix C with $|P|$ rows and $|T|$ columns. In the incidence matrix for all $t{\in}T$ and $p{\in}P$ the following hold:

$$C(p,t) = \begin{cases} -1 & \textit{if } (p,t) \in F \textit{ and } (t,p) \notin F \\ 1 & \textit{if } (t,p) \in F \textit{ and } (p,t) \notin F \\ 0 & \textit{otherwise} \end{cases} \tag{3.1}$$

The incidence matrix imposes an order on places (rows) and transitions (columns). In addition, we define consumption matrix as follows:

Definition 3.6 (**Consumption Matrix**).
A Petri net$N{=}(P, T, F)$ is denoted as a consumption matrix C' with $|P|$ rows and $|T|$ columns. In the consumption matrix for all $t{\in}T$ and $p{\in}P$ the following hold:

$$C'(p,t) = \begin{cases} -1 & \textit{if } (p,t) \in F \textit{ and } (t,p) \notin F \\ 0 & \textit{otherwise} \end{cases} \tag{3.2}$$

In the following section we associate consumption matrix with the *extended marking equation*. This matrix guarantees that enough tokens should exist before the firing of a transition. The corresponding incident matrix and consumption matrix of N_1 is shown in Fig. 3.2 and Fig. 3.3.

$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Fig. 3.2: The incidence matrix of N_1

$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Fig. 3.3: The consumption matrix of N_1

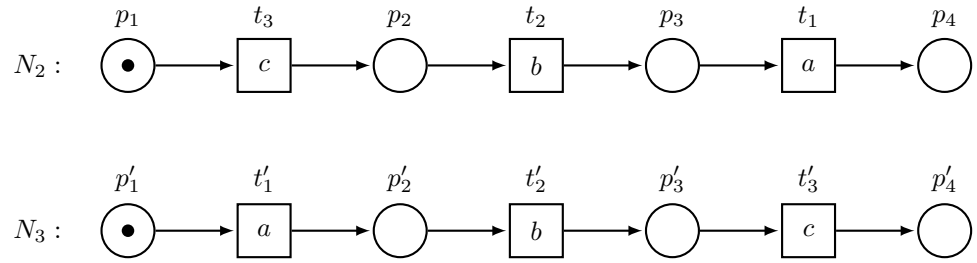
To compute alignments, we investigate a specific types of Petri net called *synchronous product net*, which is the combination of two Petri nets. In the synchronous product net, every pair of transitions with the same label compose a synchronous transition. We formalize the synchronous product net as follows:

Definition 3.7 (Synchronous Product Net).
Let $N{=}(P,T,F,\lambda)$ *and* $N'{=}(P',T',F',\lambda')$ *be two Petri nets with* $P\cap P'{=}\emptyset$ *and* $T\bigcap T'{=}\emptyset$. *The Petri net Product of* N *and* N' *is* $N{\otimes}N'{=}(P^{\otimes},T^{\otimes},F^{\otimes},\lambda^{\otimes})$ *where:*
- $P^{\otimes}{=}P{\cup}P'$,
- $T^{\otimes}{=}(T{\times}\{{\gg}\}){\cup}\{{\gg}\}{\times}T'){\cup}\{(t,t'){\in}T{\times}T' \mid \lambda(t){=}\lambda(t')\}$,
- $F^{\otimes}{=}\{(p,(t,t')){\in}P^{\otimes}{\times}T^{\otimes} \mid (p,t){\in}F \vee (p,t'){\in}F'\}{\cup}\{((t,t'),p){\in}T^{\otimes}{\times}P^{\otimes} \mid (t,p){\in}F \vee (t',p){\in}F'\}$,
- $\lambda^{\otimes} : T^{\otimes}{\rightarrow}(A{\cup}\{\tau\}{\cup}\{{\gg}\}){\times}(A{\cup}\{\tau\}{\cup}\{{\gg}\})$ *with:* $\lambda^{\otimes}(t,{\gg}){=}(\lambda(t),{\gg})$ *for* $t{\in}T$, $\lambda^{\otimes}({\gg},t'){=}({\gg},\lambda'(t'))$ *for* $t'{\in}T'$ *and* $\lambda^{\otimes}(t,t'){=}(\lambda(t),\lambda'(t'))$ *for* $t{\in}T$ *and* $t'{\in}T'$.

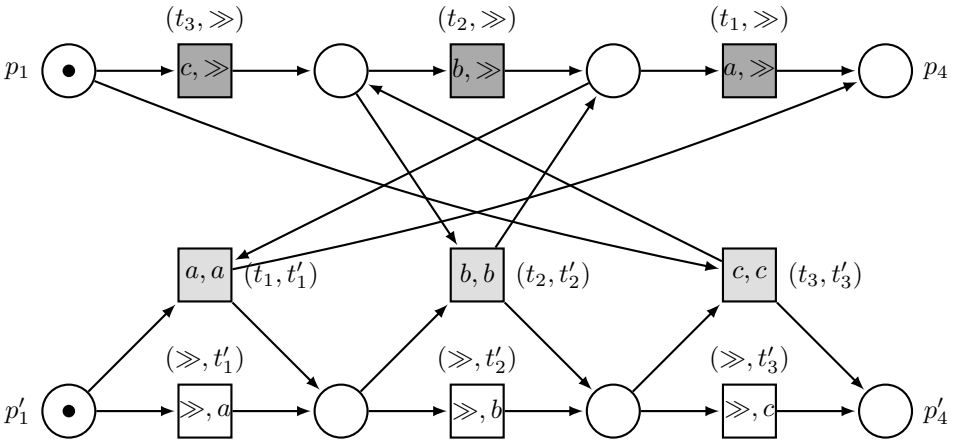
For example, given two simple Petri net N_2 and N_3 in Fig. 3.4a, we construct their synchronous product net N_4 in Fig. 3.4b.

In N_4, the top part in dark gray represents the original N_2 with transition of the form $(t,\gg)$, while the lower part of the synchronous product net in white represents N_2 with transition of the form $(\gg,t)$. The middle light gray transitions connecting to both places at the top and at bottom represents *synchronous transition*.

In this thesis, we only consider easy sound process models, i.e., the final marking is reachable from the initial marking. We formulate easy soundness as follows:



(a) Example Petri nets N_2 and N_3



(b) Synchronous product net N_4 from Petri nets N_2 and N_3

Fig. 3.4: Example of a synchronous product net

Definition 3.8 (**Easy Soundness**)**.**
Let $\sigma \in A^*$ be a trace. Let $N=(P,T,F,\lambda)$ be a labeled Petri net and let $m_i, m_f \in \mathcal{B}(P)$ denote $N's$ initial and final marking. N is easy sound *if and only if a firing sequence σ exists, such that $m_i[\sigma\rangle m_f$.*

In the following section, we introduce the concept of *alignments*, which requires the model used to be easy sound.

3.4 Alignments

Alignments are used to precisely quantify the relation between a sequence of activities in an event log and an executed sequence in a model.

In practice, we sometimes observe activities violating the specification of the model, while in other cases activities are missing whereas the model specifies they need to be executed. There are many possible reasons, such as employees mistakenly deviate from the process specification. Another possible reason is that the process specification or reference model are likely to be designed with flaws, thus the actual execution is different from expected [2]. Alignments provide detailed diagnostics about such mismatches, i.e., the deviation of behavior observed in an event log and behavior specified by a process model.

To explain the concept of alignments informally, consider the event log and the reference

model N_1 for the customer replace/repair process presented in Table 3.1 and Fig. 3.1. In Fig. 3.5, we present an alignment γ_1 for trace $\sigma_1{=}\langle a, b, d, e, g\rangle$ from case 485. The top row represents the original trace σ_1, while the bottom row corresponds to a path from the initial marking $[p_1]$ to the final marking $[p_7]$ in N_1. Every column is a *move*, and columns of the form $|\frac{a}{a}|$ denotes the observed activities matches the corresponding position in the model. In this case, trace σ_1 satisfies a direct mapping from observed activities and the execution of transitions in N_1, i.e., it aligns to the model perfectly.

γ_1 :

a	b	d	e	h
a	b	d	e	h
t_1	t_2	t_4	t_6	t_9

Fig. 3.5: Alignments for trace $\langle a, b, d, e, h\rangle$ and process model N_1

However, when the execution of activities deviates from the model, such a perfect mapping is not possible. For instance, we observe that the example trace $\sigma_2{=}\langle a, e, g\rangle$ from case 484 does not fit the behavior specified by N_1. We illustrate two possible alignments γ_2 and γ_3 in Fig. 3.6. γ_2 specifies that the execution of the d, b and h-activity are missing, i.e., the activity described by transitions t_2 and t_4 do not take place. Likewise, γ_3 specifies that an activity described by transition t_3 is missing. If we ignore the $\gg$ symbol, the top row of γ_2 and γ_3 match σ_2, and the bottom row matches a path in model N. In γ_2, columns of the form $|\frac{g}{\gg}|$ represent a *log move* that denotes observed activities in σ_2 that cannot match the corresponding position in the model, while columns of the form $|\frac{\gg}{d}|$ represent a *model move*, which denotes an activity that should take place in the model but is not observed in the corresponding position in σ_2.

γ_2 :

a	$\gg$	$\gg$	e	$\gg$	g
a	d	b	e	h	$\gg$
t_1	t_4	t_2	t_6	t_9	

γ_3 :

a	$\gg$	e	g
a	c	e	g
t_1	t_3	t_5	t_8

Fig. 3.6: Two alignments for trace $\langle a, e, g\rangle$ and process model N_1

The examples illustrate the concept of alignments as the sequence of pairs of activities in the trace and transitions in the model, in which the observed behavior is precisely related to the modeled behavior. We define the notion of alignments as follows:

Definition 3.9 (**Alignment** [16])**.**
Let $\sigma{\in}A^*$ be a trace. Let $N{=}(P, T, F, A, \lambda)$ be a labeled Petri net and let $m_i, m_f{\in}\mathcal{B}(P)$ denote the initial and final marking of N. Let $\gg{\notin}\mathcal{A}{\cup}T$. A sequence $\gamma{\in}((\mathcal{A}{\cup}\gg){\times}(T{\cup}\gg))^*$ is an alignment if:
1. $(\pi_1(\gamma))_{\downarrow\mathcal{A}}{=}\sigma$, event part equals σ.
2. $(N, m_i)\xrightarrow{(\pi_1(\gamma))_{\downarrow T}}(N, m_f)$, in which transition is in the Petri net's language.
3. $\forall(a,t){\in}\gamma(a{\neq}\gg{\wedge}t{\neq}\gg)$. Note that $(\gg, \gg)$ is not valid in an alignment.
Let $\Gamma(N, \sigma, m_i, m_f)$ denote the set of all possible alignments of Petri net N and trace σ given the initial markings m_i and the final marking m_f.

There are various alignments for trace model N_1 and $\sigma_2{=}\langle a, e, g\rangle$. We are interested in alignments that best identify mismatches between a trace and a model. Therefore, we prefer behaviors when trace and model align and penalize behaviors when they do not agree. For example, alignment γ_2 is better than γ_3 for σ_2 as less $\gg$-symbol exist, i.e., with γ_2 the trace is explained in a maximum way when referring to the model. Computing such preferred alignment is achieved by minimizing the cost function that assigns cost to the moves of alignment. The cost function for alignments is defined as follows:

Definition 3.10 (**Alignments Cost** [16])**.**
Let $\sigma{\in}A^*$ be a trace. Let $N{=}(P,T,F,\lambda)$ be a labeled Petri net and let $m_i, m_f{\in}\mathcal{B}(P)$ denote the initial and final marking of N. Let $\gg{\notin}\mathcal{A}{\cup}T$ and $c : T{\rightarrow}\mathbb{R}_{\geq 0}$. Given alignments $\gamma{\in}\Gamma(N,\sigma,m_i,m_f)$, the cost k^c of γ is defined as $k^c(\gamma){=}\Sigma_{i=1}^{\gamma}c(\gamma(i))$.

In this thesis, we define the cost of an alignment to be the total cost of all moves specified by the cost function. In the remainder of the thesis, we use the so-called unit-cost function c as follows:

1. $c(a,t){=}0$, when $a{\in}\mathcal{A}, t{\in}T$ and $\lambda(t){=}a$ or $a{=}\gg$ and $\lambda(t){=}\tau$.
2. $c(a,t){=}\infty$, when $a{\in}\mathcal{A}, t{\in}T$ and $\lambda(t){\neq}a$.
3. $c(a,t){=}1$, otherwise.

An optimal alignment is an alignment for which the total cost of sequence is minimized, i.e., it has the minimum cost among all possible alignments of the trace and the model. Given the unit-cost function, γ_1 is the optimal alignment for trace σ_1 with cost 0 as there is no other alignment that has smaller cost. Similarly, γ_2 is an optimal alignment with cost 1 for σ_2, as no other alignment exists with smaller cost. Note that uniqueness of optimal alignments is not guaranteed.

3.5 Computing Optimal Alignments

In this section we first discuss a systematic way of computing optimal alignments, i.e., solving the shortest path in the state space of the synchronous product net [31]. Then we introduce A^* search algorithm in Subsection 3.5.2, which solves the shortest path problem under the guidance of a heuristic function that underestimates the remaining costs of the optimal alignment. In Subsection 3.5.3, we present one current alignments computation algorithm.

3.5.1 From Synchronous Product Net to Alignments

As aforementioned, we compute optimal alignments by solving the shortest path problem defined on the state space of the synchronous product net of a given trace and a model. To construct such a synchronous product net for alignments computation, we first translate the trace to a linear Petri net model called *trace net*, which is defined as follows:

Definition 3.11 (**Trace net**)**.**
Let $\sigma{\in}A^*$ be a trace. We define the trace net of σ as a labeled Petri net $N{=}(P,T,F,A,\lambda)$, where $P{=}\{p_i|1{\leq}i{\leq}|\sigma|{+}1\}, T{=}\{t_i|1{\leq}i{\leq}|\sigma|\}, F{=}\{(p_i,t_i)|1{\leq}i{\leq}|\sigma|{\wedge}p_i{\in}P{\wedge}t_i{\in}T\}{\cup}\{(t_i,p_i{+}1)|1{\leq}i{\leq}|\sigma|{\wedge}p_{i+1}{\in}P{\wedge}t_i{\in}T\}$ and $\lambda{\in}T{\rightarrow}A$ with $\tau{\notin}A$.

Given σ_1=$\langle a, b, b, e, h\rangle$, we construct its trace net N_5 in Fig. 3.7. As N_5 encodes σ_1 as a sequential Petri net, we integrate it with the original model N_1 as a synchronous product net. The synchronous product net N_6 from N_1 and N_5 is depicted in Fig. 3.8.

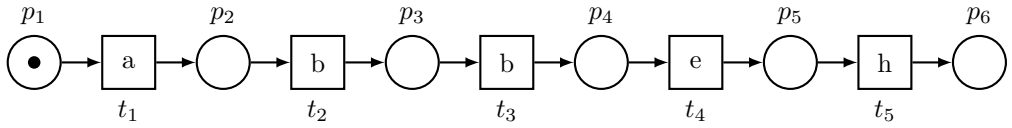


Fig. 3.7: The trace net N_5 for $\langle a, b, b, e, h\rangle$

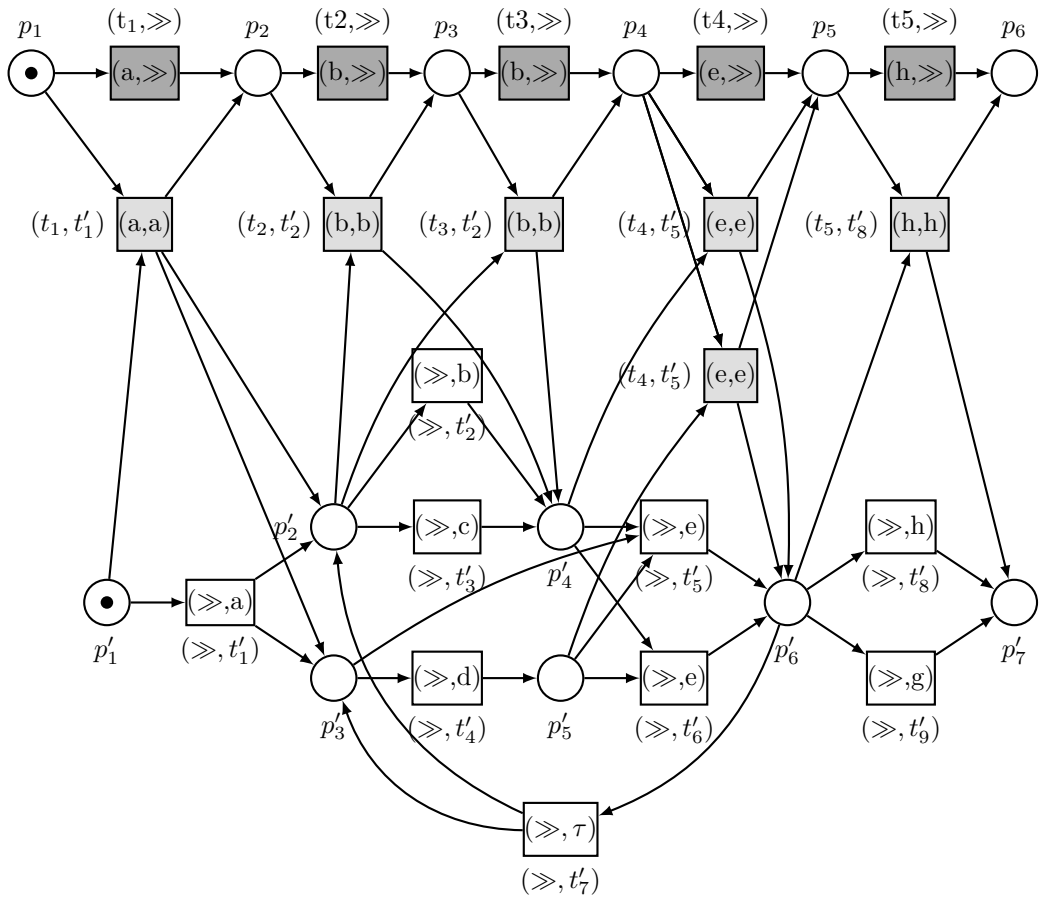


Fig. 3.8: Example Synchronous product net of N_1 and σ_1

Each transition in N_6 corresponds to a move in the alignment, i.e., transition with label of the form $(t^l, \gg)$ represents a log move, $(\gg, t^m)$ represents a model move and (t^l, t^m) represents a synchronous move.

To find an optimal alignment that minimize the cost function, we solve the shortest path problem in the state space of a synchronous product net constructed from the given trace and model.

3.5.2 A^* Search Algorithm

In this section, we review some de-facto algorithms that solve the shortest path problem for a weighted graph.

Dijkstra's Algorithm [32] is the classic algorithm for solving the shortest path problem in a weighted graph. It starts from the initial vertex, then repeatedly examines the closest not-yet-examined vertex, adding its neighbors to the set of neighbors to be examined. It expands outwards from the starting vertex until it reaches the goal. As long as all edges have a non-negative cost, Dijkstra's algorithm is guaranteed to find the shortest path from the start to the goal.

Greedy best-first-search takes advantage of the so-called heuristic [33] to decide which neighbor is the most promising vertex and should be explored next. Thus, instead of selecting the vertex closest to the starting vertex like Dijkstra's Algorithm, it selects the vertex that is estimated to be the closest to the goal with a heuristic. The search maintains two sets, i.e. open set and closed set. The open set keeps vertex that have not been expanded, while the closed set stores already expanded vertices. During each iteration, vertex with the lowest heuristic value is selected and placed to the closed set, and the algorithm explores all its successors. The search continues until the goal is found.

In our thesis, we focus on the A^*-based search algorithm [34], which not only extends Dijkstra's algorithm but also incorporates the advantage of greedy best-first-search. The algorithm also maintains open set for not yet expanded vertices and closed set for already expanded vertices. During search, it selects the most promising vertex in the graph to visit next. What makes A^* better is that for each vertex selected, it uses a function $f(n)$ that considers an estimate of the total cost of the path using that vertex n. i.e., the path selected minimizes $f(n)=g(n)+h(n)$, where $g(n)$ is the cost from start vertex to n and $h(n)$ is an estimated heuristic that denotes the cost of remaining path from n to goal. Thus, the vertex selected is based on the distance to the starting vertex as well as the estimated distance to the goal. After reaching the goal, the search terminates. We compare the three different search algorithms in Table 3.2.

Table 3.2: Summary of three different shortest path search algorithm

Name	**Method description**	**Features**
Dijkstra	1. Explore the closest to starting vertex yet unexplored vertex.	1. Guarantee optimality. 2. Run slower.
Greedy BFS	1. Explore the closest to target vertex yet unexplored vertex.	1. Run quicker. 2. Cannot guarantee optimality.
A-star(A^*)	1. Explore the vertex considering both the distance to start vertex and the distance to target vertex.	1. Guarantee optimality if heuristic is admissible.

A search algorithm is admissible if it is guaranteed to return an optimal solution. Thus, if the heuristic function used by A^* algorithm is admissible, i.e., the heuristic $h(n)$ for current vertex n is always less than or equal to the actual distance to target vertex, then A^* algorithm is admissible.

As for the computation of optimal alignment, the state space with initial and final marking

represents the graph to be traversed, and the optimal alignments correspond to the shortest path from the initial marking to final marking in the state space. To apply A^* search algorithm, we need to define a proper heuristic function.

3.5.3 Underestimation with Marking Equation

When applying A^* search for the shortest path problem, a heuristic function is needed to guide the search. Thus we formally define the underestimation function in the context of alignments computation as follows:

Definition 3.12 (**Underestimation Function**).
Let $N{=}(P,T,F,A,\lambda)$ be a labeled Petri net and c: $T{\rightarrow}\mathbb{R}_{\geq 0}$ be a cost function. $h : \mathcal{B}(P){\rightarrow}\mathbb{R}_{\geq 0}$ is an underestimation function if and only if for all $m{\in}\mathcal{B}(P)$ and $\sigma{\in}A^*$ with $m[\sigma\rangle m'$, $h(m){\leq}c(\sigma)$holds.

We apply marking equation for underestimation function, since it exploits the structure of a Petri net with linear algebra. We define marking equation as follows:

Definition 3.13 (**Marking Equation**).
Let C be the incidence matrix of Petri net. If a sequence σ exists, such that $m[\sigma\rangle m'$, the marking equation states: $\vec{m} + C \cdot \vec{\sigma}{=}\vec{m'}$.

Consider Petri net N_1 and its incidence matrix, we represent the initial marking $[p_1]$ with column vector $\vec{m_i}^{\mathsf{T}}{=}(1,0,0,0,0,0,0)$, which means we assign value 1 to place $p1$ and 0 to other places. Likewise, the column vector for final marking $[p_7]$ is $\vec{m_f}^{\mathsf{T}}{=}(0,0,0,0,0,0,1)$, which means only place p_7 is assigned with value 1. We assume the order of transition is $\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9\}$, thus Parikh vector $\vec{\sigma}^{\mathsf{T}}{=}(1,1,0,0,0,0,0,0,0)$ indicates transitions t_1 and t_2 are fired. If the final marking is reachable from the initial marking, the marking equation present in Fig. 3.9 has at least one solution.

$$\begin{pmatrix}1\\0\\0\\0\\0\\0\\0\end{pmatrix} + \begin{pmatrix}-1&0&0&0&0&0&0&0&0\\1&-1&-1&0&0&0&0&0&0\\1&0&0&-1&0&-1&1&0&0\\0&1&1&0&-1&-1&0&0&0\\0&0&0&1&0&-1&0&0&0\\0&0&0&0&1&1&-1&-1&-1\\0&0&0&0&0&0&0&1&1\end{pmatrix} \cdot \begin{pmatrix}a\\b\\c\\d\\e\\e\\\tau\\g\\h\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\0\\1\end{pmatrix}$$

Fig. 3.9: Marking equation example

Since we assume the model used is easy sound, the reachability of the final marking is guaranteed. Trace $\sigma_1{=}\langle a,b,d,e,g\rangle$ is a realizable firing sequence from the initial marking to the final marking. We assign value 1 to transitions $t_1,t_2,t_4,t5,t_9$, and get a Parikh vector $\vec{1}^{\mathsf{T}}{=}(1,1,0,1,1,0,0,0,1,0)$, which is a solution to the marking equation.

On the other hand, if the marking equation has no solution, no firing sequence exists from marking m to marking m'. Also, even if a solution exists, the corresponding firing sequence does not necessarily correspond to a realizable firing sequence.

Definition 3.14 (**ILP-based Underestimation Function**). .
Let $N{=}(P,T,F,A,\lambda)$ be a labeled Petri net with incidence matrix C and let c: $T{\to}\mathbb{R}_{\geq 0}$ be a cost function. Let $\vec{c}$ be a column vector assigning cost values for each transition. We define $h^{ILP} : \mathcal{B}(P){\to}\mathbb{R}_{\geq 0}$ as an ILP based underestimation, so that for each marking $m{\in}\mathcal{B}(P)$, it holds that $h^{ILP}(m){=}\vec{c}^{\mathsf{T}} \cdot \vec{\sigma}$ where $\vec{\sigma}$ is the solution to:
minimize

$$\vec{c}^{\mathsf{T}} \cdot \vec{\sigma}$$

subject to

$$\vec{m} + C \cdot \vec{\sigma} = \vec{m}_f$$

Consider Petri net N_1, its underestimation function for initial marking is written as:

$$\begin{array}{ll} \textit{minimize} & 1 \cdot a + 1 \cdot b + 1 \cdot c + 1 \cdot d + 1 \cdot g \\ \textit{subject to} & 1 - a = 0 \\ & a - b - c = 0 \\ & a - d - e + \tau = 0 \\ & b + c - d - e = 0 \\ & d - e = 0 \\ & e + e - \tau - g - h = 0 \\ & g + h = 1 \end{array}$$

Underestimation function is used to compute heuristic to guide the search, and the solution vector $\vec{\sigma}$ is used to derive a solution vector for subsequent markings [16]. As long as the corresponding row in solution vector allows the firing of the transition from the previous marking to the subsequent marking, $\vec{\sigma}$ is reused to derive feasible heuristic, otherwise not. Thus we discuss the feasibility of the heuristic as follows:

Definition 3.15 (**Feasibility of Heuristic**).
Let $N{=}((P,T,F,A,\lambda),m_i,m_f)$ be a labeled Petri net and c: $T{\to}\mathbb{R}_{\geq 0}$ be a cost function. Let m, m' be two markings and $\vec{\sigma}$ a solution for m that minimizes ILP function defined in Definition 3.14. Let $t{\in}T$ be a transition such that $m[t\rangle m'$. If $\vec{\sigma}(t) \geq 0$, $h(m'){=}h(m) - c(t)$ is feasible with with solution $\vec{\sigma'}{=}\vec{\sigma} - \vec{1_t}$, where $\vec{1_t}$ is a vector with 1 at the row corresponding to t and 0 to other rows. If $x(t) < 0$, then the heuristic for m' is infeasible.

In the remainder of the thesis, we refer to the markings with feasible heuristic as feasible markings whereas those without a feasible solution as infeasible markings. We exemplify the feasibility of solution vector in Fig. 3.10, assume the solution vector for marking m_j is $\vec{\sigma_j}{=}(0,1,2,1,0,1,0)$. In $\vec{\sigma_j}$, we assume transition t_k corresponds to row 0, transition t_l to 1 and transition t_n to 2. To explore further from $[p_5]$, we reuse $\vec{\sigma_j}$ to derive heuristic for subsequent marking m_k, m_l and m_n. However, subtracting $\vec{1_{t_j}}$ from $\vec{\sigma_j}$ leads to negative value at row 0. Thus, the heuristic for m_k is infeasible. In contrast, as the rows corresponding to transitions t_l and t_n are greater than 0, the heuristic for m_l and m_n are feasible. We refer to m_k as infeasible marking and m_l and m_n as feasible markings.

The state space traversal algorithm proposed in [16] favors feasible markings and reuses previously obtained solution vectors. We illustrate the flow chart of this algorithm in Fig. 3.11.

The input are a synchronous product net and a cost function. At first the closed set and open set are initialized as an empty set and a queue, respectively. After computing the h-value and solution vector for initial marking, it is inserted into the open set. As long as

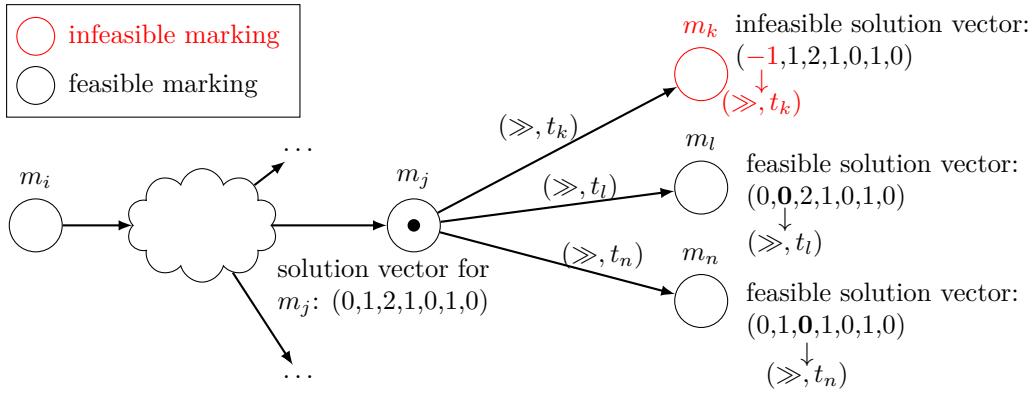


Fig. 3.10: Example of feasible/infeasible markings

the open set is not empty, the algorithm iteratively selects the most promising marking m with minimal f-value. In case m is the final marking, we construct the optimal alignment. Otherwise, we check the feasibility of solution vector for m. If m is infeasible, we calculate a new feasible heuristic for it using the ILP-based underestimation function. Otherwise, m is inserted into the closed set and we explore subsequent markings of m. If the newly explored marking is in closed set, we ignore it. Otherwise, it is inserted into the open set. Specifically, if the newly explored marking is already in the open set and the alternative sequence leading to it is shorter, we update its g-value. In the context of this algorithm, g-value refers to the cost of sequence. The algorithm iterates recursively until the final marking is reached.

Unfortunately, the basic A^* algorithm using (I)LP based underestimation function performs poorly in certain cases. Consider the example synchronous product net N_4 in Fig. 3.4b, the solution to the underestimation function suggests that through sequence $\langle (a,a),(b,b),(c,c)\rangle$ the final marking is reachable from the initial marking with zero cost. However, it is not possible because none of these transitions are enabled by the initial marking. The optimal alignment is $\langle (a,\gg),(\gg,a),(b,b),(\gg,c),(c,\gg)\rangle$ with a cost of 4. The reason why the marking equation fails is that the events a, c in N_2 and N_3 are swapped, thus the solution vector which minimizes the underestimation function does not correspond to a realizable firing sequence. To improve the quality of the heuristic, we define the extended marking equation and introduce a *split-point-based algorithm* in the next section.

3.6 Split-Point-Based Search

In this section, we present the split-point-based search that computes optimal alignments based on the *extended marking equation* [4]. We formalize extended marking equation as follows:

Definition 3.16 (**Extended Marking Equation** [4])**.**
Let $N{=}(P,T,F,\lambda)$ be a Petri net, C and C' be its incidence matrix and consumption matrix. Let $\sigma_1,\sigma_2{\in}T^*$ be two firing sequences, such that firing sequence σ_2 starts with transition t. Let m_1,m_2,m_3 be three markings, so that $m_1[\sigma_1\rangle m_2$ and $m_2[\sigma_2\rangle m_3$ hold.

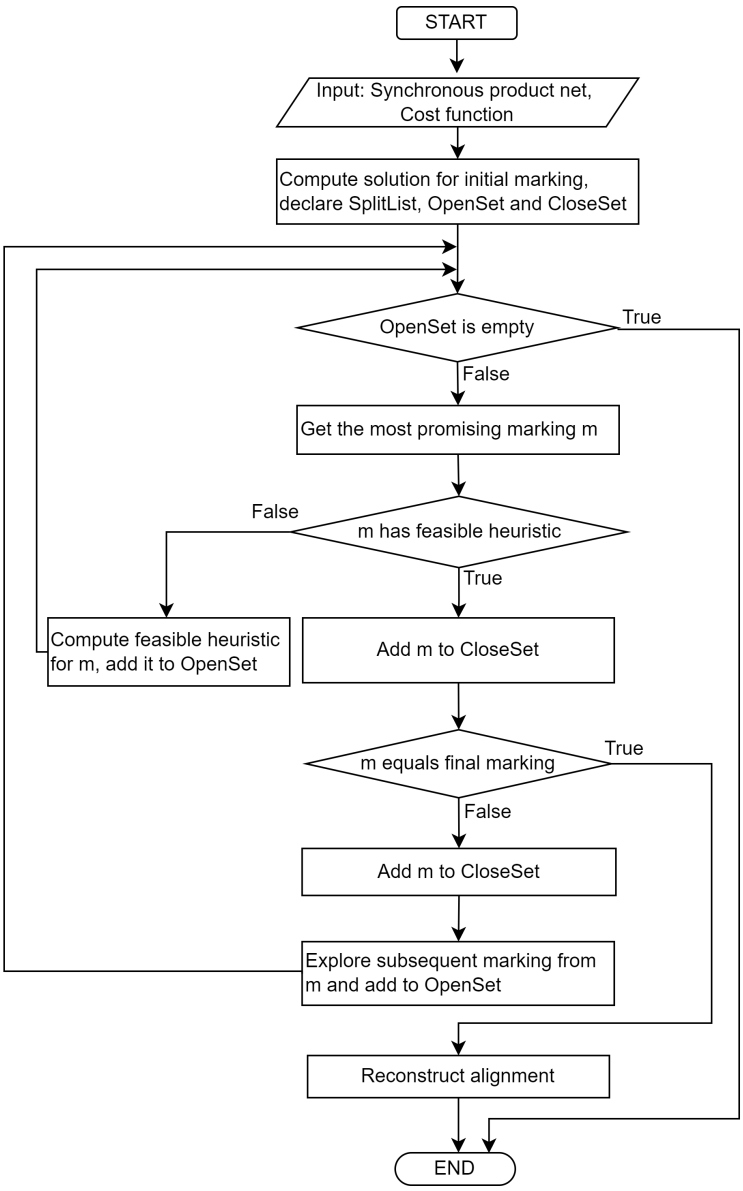


Fig. 3.11: The flowchart of the basic alignments computation algorithm in [16]

The extended marking equation is defined as:

$$\begin{cases} \vec{m_1} + C \cdot (\vec{\sigma_1} + \vec{\sigma_2}) = \vec{m_3} \\ \vec{m_1} + C \cdot \vec{\sigma_1} + C' \cdot \vec{1_t} \geq 0 \end{cases} \tag{3.3}$$

The first equation is essentially the same as marking equation defined in Definition 3.13, while the second one emphasizes an enabling condition, i.e., transition t is enabled only when enough tokens exist in each of its input place. After firing sequence σ_1 we reach marking m_2 from m_1. After firing the first transition t from $_2$ and consuming necessary tokens, the number of tokens in any place remains non-negative. Note that the solutions for σ_1 and σ_2 do not always correspond to realizable firing sequence, even if they satisfy the extended marking equation.

Considering that the extended marking equation imposes more restrictions, we take advantage of it for the underestimation function by identifying the split point that exists between σ_1 and σ_2.

Definition 3.17 (**Underestimation Function with k Sub-traces for the initial marking**).
Let $c : T \to \mathbb{R}_{\geq 0}$ be a cost function, σ be a trace, $SN=(P, T, F, \lambda)$ be a synchronous product net constructed from the trace net of σ and a model net, and m_i, m_f be the initial and final markings. We split original σ into $\sigma_0 \cdot \ldots \cdot \sigma_k$, i.e., k+1 pieces of non-empty subtraces. We define $h_{m_i}=\vec{c}\cdot\Sigma_{0\leq a\leq k}\vec{x_a} + \vec{c}\cdot\Sigma_{0\leq a<k}\vec{y_a}$ where $\vec{x_a}$ and $\vec{y_a}$ are the solutions to:
minimize

$$\vec{c}^{\mathsf{T}} \cdot \Sigma_{0\leq a\leq k}\vec{x_a} + \vec{c}^{\mathsf{T}} \cdot \Sigma_{0\leq a\leq k}\vec{y_a}$$

subject to

$$\vec{m_i} + C \cdot \vec{x_0} + C \cdot \Sigma_{1\leq b\leq k}(\vec{x_b} + \vec{y_b}) + C' \cdot \vec{y_a} \geq \vec{0} \tag{1}$$

$$\vec{m_i} + C \cdot \vec{x_0} + C \cdot \Sigma_{1\leq b<a}(\vec{x_a} + \vec{y_a}) = \vec{m_f} \tag{2}$$

$$\forall_{1\leq a\leq k}\ \forall_{t\in T}\ \vec{y_a}(t) \in \{0, 1\} \tag{3}$$

$$\forall_{1\leq a\leq k}\ \forall_{(t_m,t_l)\in T}\ \textit{with } t_l \neq t_{(1+\Sigma_{1\leq b<a}|\sigma_b|)}\vec{y_a}((t_m, t_l)) = 0 \tag{4}$$

$$\forall_{0\leq a\leq k}\ \vec{1} \cdot \vec{y_a} = 1 \tag{5}$$

$$\forall_{0\leq a\leq k}\ \forall_{t\in T}\ \vec{x_a}(t) \in \mathbb{N} \tag{6}$$

We assume that at least one optimal alignment exists in the model given the model is easy sound, thus this equation is guaranteed to provide a solution. Given such optimal alignment $\gamma \in T^*$ for a trace σ, we divide it into $k+1 (k \leq |\sigma|)$ sub-parts such that $\gamma = \gamma_0 \cdot \ldots \cdot \gamma_k$. Each sub-parts γ_a consists of x_a and y_a, where y_a of the form (t_l, t) or $(t_l, \gg)$ matches one event t_l in the trace, and x_a corresponds to subsequent transitions firing in arbitrary order. This implies also σ_i starts with one transition that moves tokens in the original trace model [4]. Equation 1 implies the original marking equation, which implies that final marking is reachable from the initial marking if we fire all transitions in every x_a and y_a. Equation 2 is based on the extended marking equation that guarantees after firing the previous subtrace from σ_0 to σ_{a-1}, there are sufficient tokens available for the firing of the first transition $\vec{y_a}$ in σ_a. Equation 3, Equation 4 and Equation 5 state that variable $\vec{y_a}$ is the first transition in each subtrace, and the solution vector contains only one element that equals 1, while other elements are 0. Specifically, the element with value 1 corresponds to the starting transition of subtrace σ_a from the synchronous product net. Equation 6 states that variable x_a have any other transition firing sequences in the corresponding subtrace.

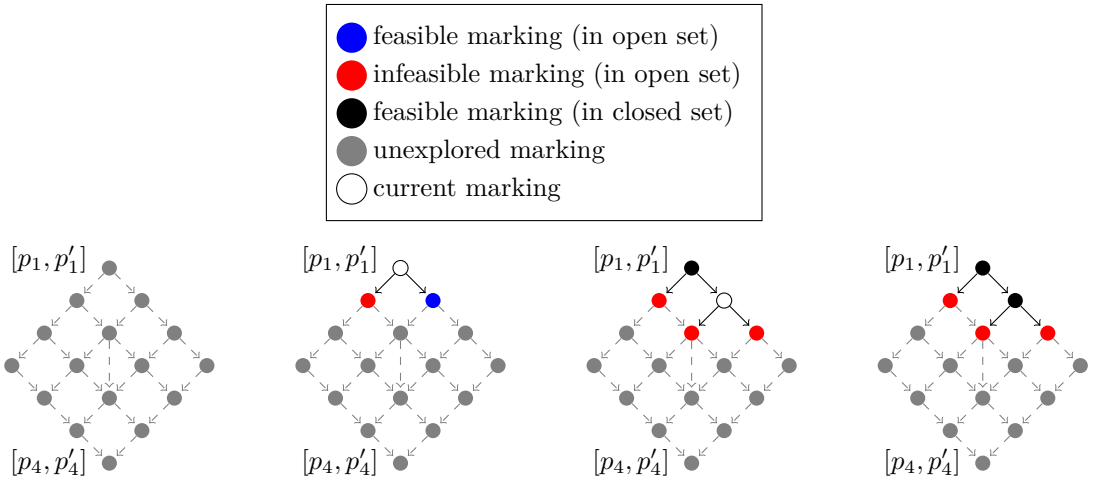
Definition 3.17 provides an underestimation function for the total cost of optimal alignments for the initial marking. Given the extended marking equation, we transform the problem of computing optimal alignments for the original trace into computing optimal alignments for k smaller pieces of predefined subtraces.

Definition 3.18 (**Underestimation with k+1 Sub-traces** [4]).
Let σ be a trace, $SN=(P, T, F, \lambda)$ be a synchronous product net constructed from the trace net of σ and a model net. Let $m \in \mathcal{B}(P)$ be a marking in which l events of trace σ are explained and let $\sigma=\sigma_0 \cdot \ldots \cdot \sigma_k$ be a division of the trace into k+1 non-empty subtraces, such that the first l events are in σ_0, namely $k \leq |\sigma|-l$. We define the underestimation function $h^{ILP,k} : \mathcal{B}(P) \to \mathbb{R}_{\geq 0}$ to follow Definition 3.17.

For example, given a trace $\sigma{=}\langle a, b, c\rangle$, we split σ into $\sigma_0 \cdot \sigma_1$ where $\sigma_0{=}\langle a\rangle$ and $\sigma_1{=}\langle b, c\rangle$, thus we solve the underestimation function with 2 sub-problems.

We exemplify the way split-point-based approach works for the state space search of Fig. 3.4b in Fig. 3.12, Fig. 3.13 and Fig. 3.14. In each state space, different colors indicates different markings. We mark markings in gray if they are not yet explored. Blue markings represent they have a feasible heuristic and red ones do not. The white marking refers to the current marking inspected by the algorithm, and black markings mean they are already in closed set. As for the arcs, the gray dashed ones represent unexplored arcs and black solid ones represent already traversed arcs. In the remainder of the thesis, we adopt this color scheme.

In Fig. 3.12, we start the search from an unexplored state space. At step 2, the initial marking is used as the current marking (in white). One of its subsequent markings (in blue) has a feasible heuristic, the other one (in red) does not have feasible solution. The algorithm stores them in open set. At step 3, the initial marking is removed from the open set and stored in closed set (in black), which means it has been explored. Its feasible successor (in white) is used as current marking to continue the search. At step 4, the search gets stuck as none of the markings in open set have a feasible heuristic.



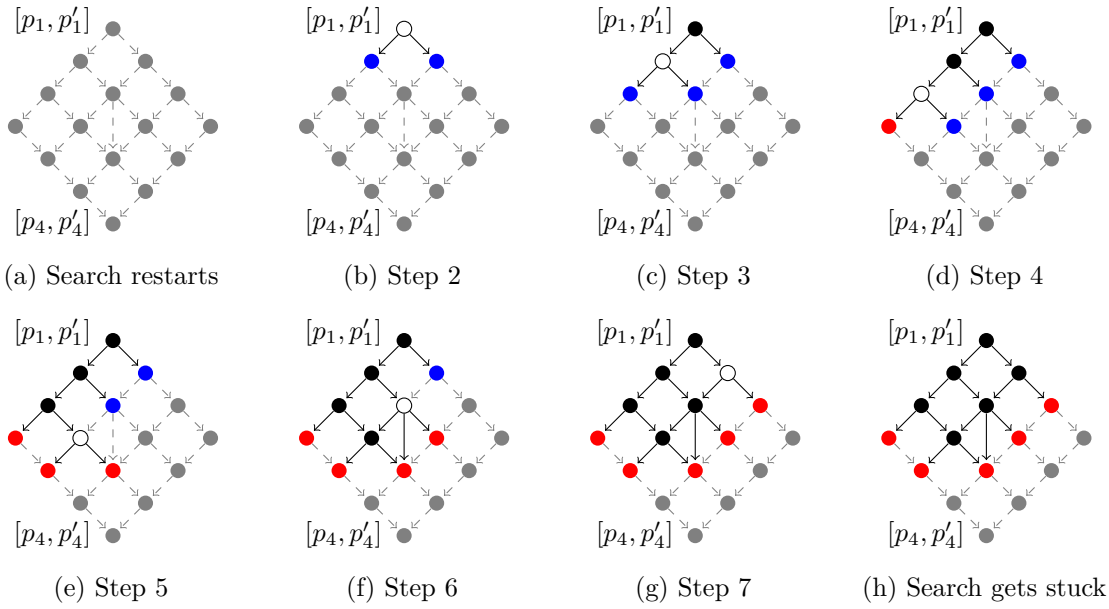
(a) Search starts (b) Step 2 (c) Step 3 (d) Search gets stuck

Fig. 3.12: The first round of search for the state space of Fig. 3.4b

In this case, split-point-based technique introduces a split point, and re-computes a new heuristic for the initial marking using the extended marking equation. Thus, the algorithm restarts the search from scratch as present in Fig. 3.13,.

During the second round of search, the algorithm explores the state space towards a different direction under the guidance of initial heuristic and solution vector computed. Likewise, from step 2 to step 7, the algorithm iteratively selects the most promising marking and expands from it. Although the search gets stuck again before reaching the final, more markings have been explored than the first round. In the third round depicted in Fig. 3.14, the algorithm successfully reaches the final marking under the guidance of another initial heuristic.

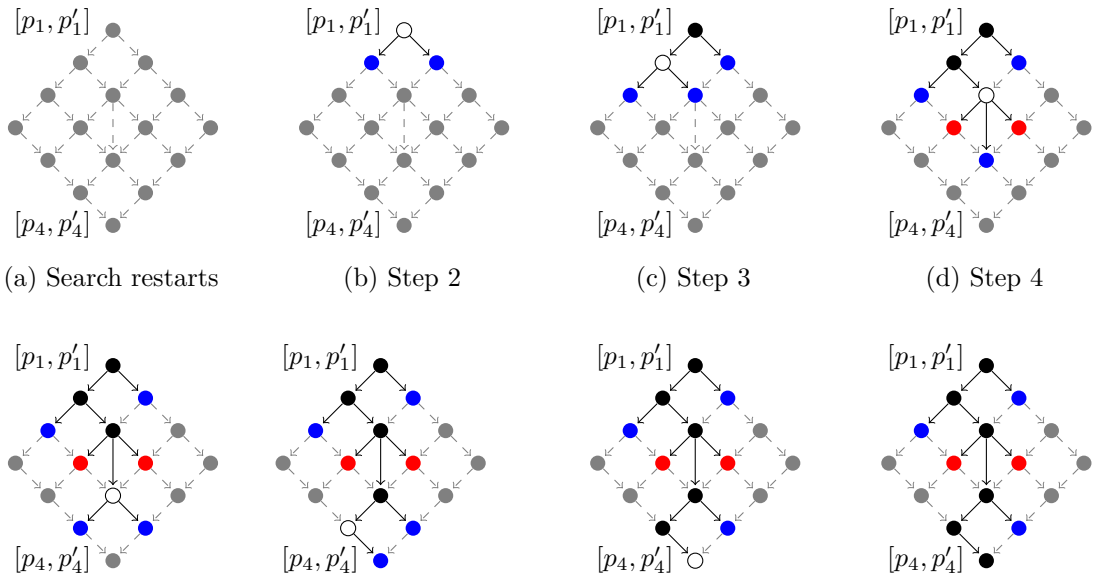
We present the flow chart of this algorithm in Fig. 3.15. It takes a synchronous product net



(a) Search restarts (b) Step 2 (c) Step 3 (d) Step 4

(e) Step 5 (f) Step 6 (g) Step 7 (h) Search gets stuck

Fig. 3.13: The second round of search for the state-space of Fig. 3.4b



(a) Search restarts (b) Step 2 (c) Step 3 (d) Step 4

(e) Step 5 (f) Step 6 (g) Step 7 (h) Reach final marking

Fig. 3.14: The third round of search for the state space of Fig. 3.4b

and a cost function as input. After initializing the open set and closed set, it calculates the heuristic for the initial marking. Then the algorithm iteratively selects the most promising marking to explore. If the marking equals the final marking, the search ends and an alignment is constructed. Otherwise, we check the feasibility of the marking. If the marking is infeasible, the search reaches a point where the original computed solution for initial marking corresponds to a non-realizable firing sequence. In this case, the search gets stuck and the algorithm further splits the original trace with a new split point, which

is determined based on the maximum index of event explained by any marking reached so far. After computing the extended marking equation, a new heuristic for the initial marking is computed and can be used to start another round of search. However, if no new split point is found when the search gets stuck, the algorithm turns to a regular A^* search, i.e., it calculates an exact solution for the most promising infeasible marking and continues the search from it. In general, the split-point-based algorithm incrementally improves the underestimation function with more split points and maximizes the reuse of the previously computed solution vector.

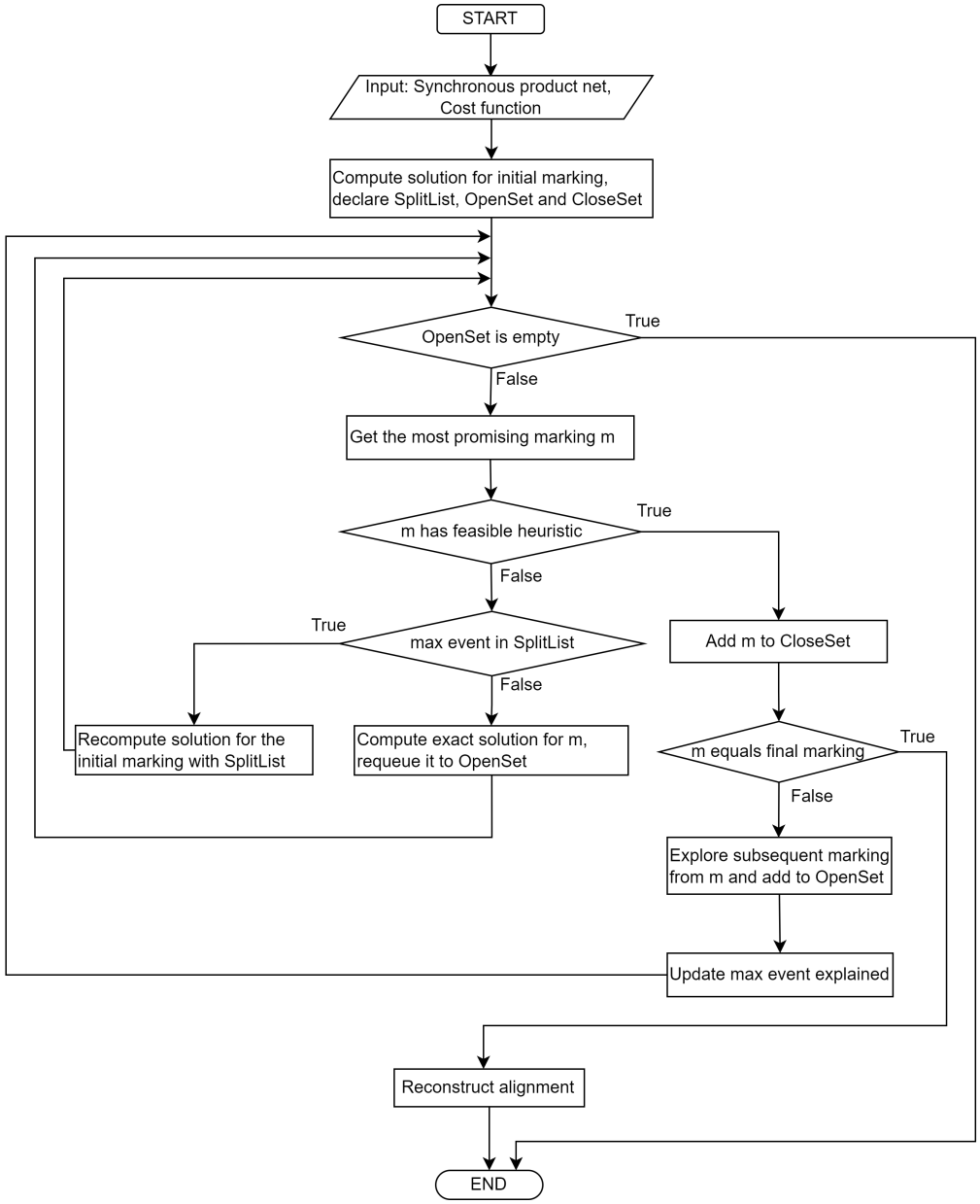


Fig. 3.15: The flowchart of the incremental alignments computation algorithm in [4]

Chapter 4

Method

In this chapter, we propose a caching strategy that enhances the original split-point-based search algorithm. In Section 4.1, we motivate our approach by analyzing the drawback of split-point-based approach. In Section 4.2, we provide an overview of the caching strategy. In Section 4.3, we elaborate its implementation. To deal with the side effect of the caching strategy, we introduce a sequence propagation technique in Section 4.4. Finally in Section 4.5, we discuss a way to find a trade off between always and never using caching strategy.

4.1 Analysis of Split-point-based Algorithm

While split-point-based algorithm provides a better initial heuristic estimation, it discards all information gathered during the previous round of search and restart from scratch. When the search gets stuck and a new split point is added, all markings in both open set and closed set get erased.

We investigate the drawback of erasing previously obtained results in Fig. 4.1, which shows the number of processing times for each marking during the search discussed in Fig. 3.12 to Fig. 3.14.

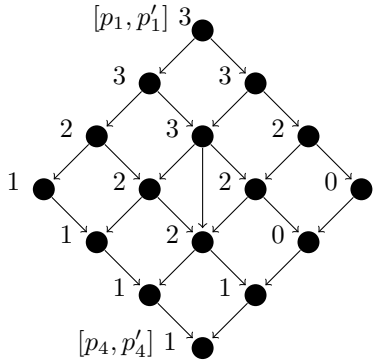
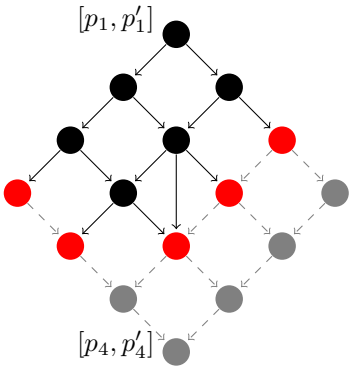


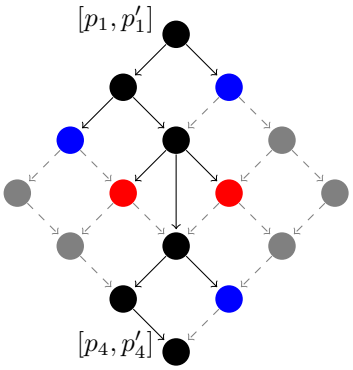
Fig. 4.1: The number of time to process each marking in the state space of N_4

There are 16 markings in the state space, but the total number of explorations times for

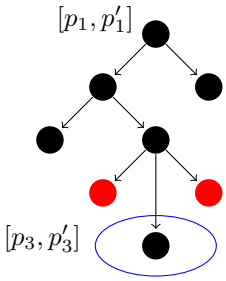
all marking sums up to 27 times. For example, the initial marking is explored three times during the search, as well as its two subsequent markings. As shown in Fig. 4.2a, after the second round of search, there are 6 feasible markings in the closed set and 5 infeasible markings in the open set. After the third round of search, the final marking is reached as present in Fig. 4.2b. We show the overlapping part of the state space in Fig. 4.2c, where the markings are explored both during the second and third round of search. Although the black marking marked in blue circle was infeasible in Fig. 4.2a, it becomes feasible during the third round of search.



(a) Search gets stuck during the second round of search



(b) Reach final marking in the third round of search



(c) The overlapping part of the state space for the second and third round of search

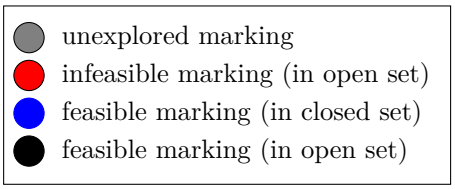


Fig. 4.2: Marking $[p_3, p_3']$ becomes feasible after recomputing initial heuristic

Thus, instead of completely restarting from scratch, the infeasible markings from previous round of search is a potential shortcut to take, as the search can continue from them. In the following chapter, we introduce a cache set to store all infeasible markings during the search. We seek to improve the state space traversal efficiency by reusing previously obtained results and continuing the search from markings in the cache set.

4.2 Caching Strategy Overview

In this section, we provide a general overview of the way naive caching strategy works.

Unlike the original split-point-based search that stores all newly explored markings in the open set, *caching strategy* only stores feasible markings in the open set, while leaving the infeasible markings in a so-called *cache set*. If the search gets stuck and a new initial heuris-

tic is computed, instead of restarting from scratch like the split-point-based approach, we examine all infeasible markings in the cache set and check whether it is possible to continue the search from them. This is achievable if we keep track of the Parikh vector of the previous sequence for each marking. When examining the cache set, we iteratively select an infeasible marking and derive a new solution vector by subtracting its Parikh vector from the initial solution. Then we verify whether all rows in the newly derived solution vectors are feasible as defined in Definition 3.15. If it is the case, the selected infeasible marking has a feasible heuristic and we insert it into the open set. Otherwise, the marking remains in the cache set.

We exemplify the way to reuse infeasible markings in Fig. 4.3 to Fig. 4.5, which are based on the state space search presented in Fig. 3.13 and Fig. 3.14. Note that the synchronous product net has 9 transitions, thus the solution vector for the initial marking and the Parikh vector of any sequence are column vectors with 9 rows. At the beginning of the search, the solution vector for initial marking $[p_1, p_1']$ is (0,1,0,0,0,0,1,0,1).

When the search gets stuck at the first round of search, we present part of the state space explored is present in Fig. 4.3. $[p_1, p_1']$, $[p_1, p_2']$ are in closed set, infeasible markings $[p_2, p_1']$, $[p_2, p_2']$ and $[p_1, p_3']$ are in open set. The Parikh vectors for three infeasible markings are (1,0,0,0,0,0,0,0,0), (0,1,0,0,0,0,0,1,0) and (0,1,0,0,0,1,0,0,0).

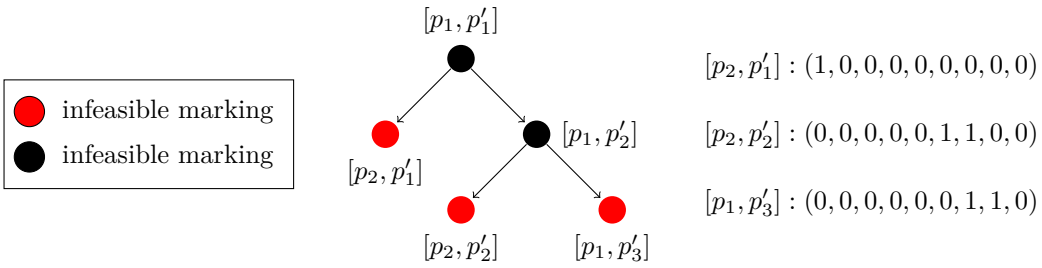
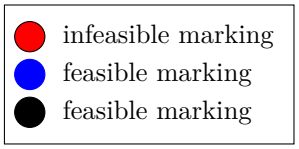


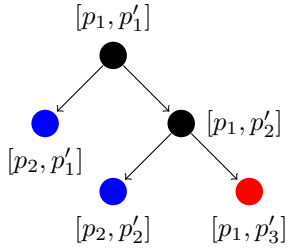
Fig. 4.3: The Parikh vectors of infeasible markings after the first round of search

After computing a new initial solution vector (1,0,0,0,0,1,1,0,0), we subtract the Parikh vector stored in $[p_2, p_1']$, $[p_2, p_2']$ and $[p_1, p_3']$ to derive new solution for them. The result in Fig. 4.4 shows that two previous infeasible markings $[p_2, p_1']$ and $[p_2, p_2']$ have a feasible solution vector after subtraction, while marking $[p_1, p_3']$ remains infeasible as it contains negative value.

Likewise, when the search gets stuck in the second round of search in Fig. 4.5, there are 5 infeasible markings. Then in Fig. 4.6, we derive new solution vectors for them by subtracting the Parikh vector from the new initial solution vector (1,0,1,0,1,0,1,0,1). In this case, marking $[p_3, p_3']$ becomes feasible, while the rest remains infeasible. If we continue the search from $[p_3, p_3']$, the final marking is reachable in two steps. In comparison, the original split-point-based algorithm restarts from the initial marking and takes 7 steps before reaching the final marking.

We refer to the infeasible markings in cache set as the search frontier, as the algorithm does not expand further from it. When the search gets stuck, checking infeasible markings in the cache set enables the search to continue from the frontier of the previous search.



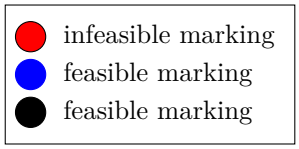


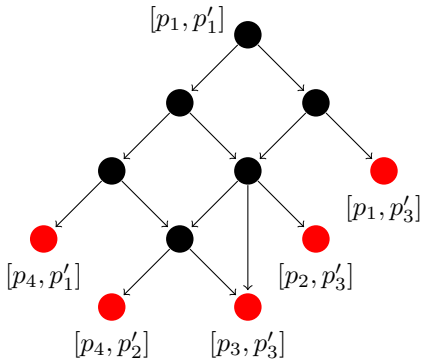
$$[p_2, p'_1] : (1,0,0,0,0,1,1,0,0)$$
$$-(1,0,0,0,0,0,0,0,0)$$
$$= (0,0,0,0,0,1,1,0,0)$$

$$[p_2, p'_2] : (1,0,0,0,0,1,1,0,0)$$
$$-(0,0,0,0,0,1,1,0,0)$$
$$= (1,0,0,0,0,0,0,0,0)$$

$$[p_1, p'_3] : (1,0,0,0,0,1,1,0,0)$$
$$-(0,0,0,0,0,0,1,1,0)$$
$$= (1,0,0,0,0,1,0,-1,0)$$

Fig. 4.4: Computing new solution vector for infeasible markings with new initial solution





$$[p_1, p'_3] : (0,0,0,0,0,0,1,1,0)$$

$$[p_4, p'_1] : (1,1,1,0,0,0,0,0,0)$$

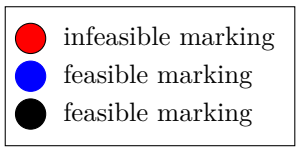
$$[p_2, p'_3] : (1,0,0,0,0,0,1,1,0)$$

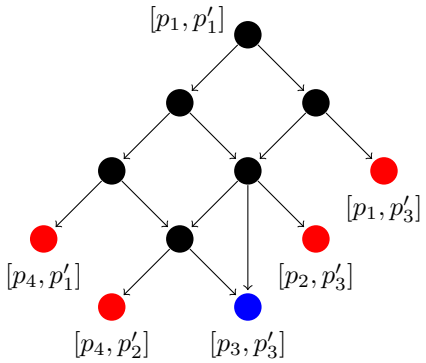
$$[p_4, p'_2] : (1,1,1,0,0,0,1,0,0)$$

$$[p_3, p'_3] : (1,0,0,0,1,0,1,0,0)$$

Fig. 4.5: The Parikh vectors of infeasible markings after the second round of search

$$[p_1, p'_3] : (1,0,1,0,1,0,1,0,1)$$
$$-(0,0,0,0,0,0,1,1,0)$$
$$= (1,0,1,0,1,0,0,-1,1)$$





$$[p_4, p'_1] : (1,0,1,0,1,0,1,0,1)$$
$$-(1,1,1,0,0,0,0,0,0)$$
$$= (1,-1,0,0,1,0,1,0,1)$$

$$[p_2, p'_3] : (1,0,1,0,1,0,1,0,1)$$
$$-(1,0,0,0,0,0,1,1,0)$$
$$= (0,0,1,0,1,0,0,-1,1)$$

$$[p_4, p'_2] : (1,0,1,0,1,0,1,0,1)$$
$$-(1,1,1,0,0,0,1,0,0)$$
$$= (0,-1,0,0,1,0,0,0,1)$$

$$[p_3, p'_3] : (1,0,1,0,1,0,1,0,1)$$
$$-(1,0,0,0,1,0,1,0,0)$$
$$= (0,0,1,0,0,0,0,0,1)$$

Fig. 4.6: Computing new solution vector for infeasible markings with new initial solution

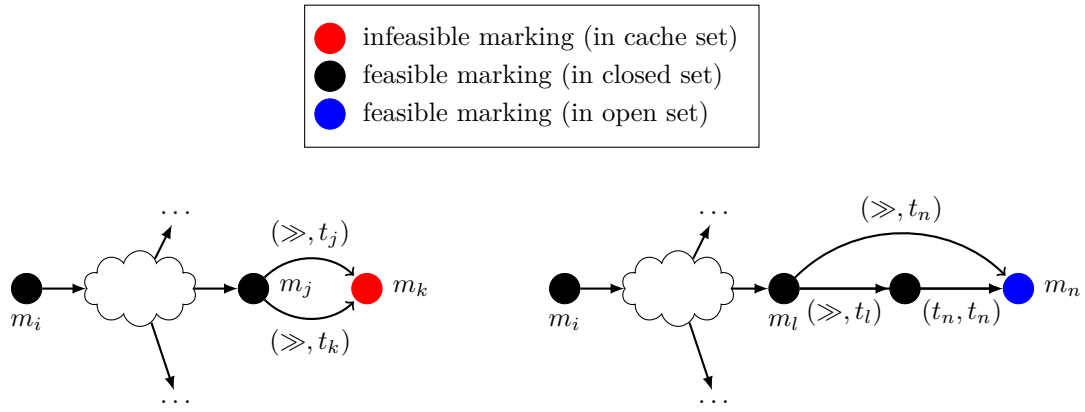
4.3 Naive Caching Strategy

In this section, we first discuss various cases where the Parikh vector of a firing sequence should be kept or pruned for a marking, then we formally introduce the caching strategy that extends the split-point-based approach.

At first, we discuss cases where multiple different Parikh vectors exist for a reachable marking. As aforementioned, it is possible for a marking to be reached multiple times with different sequences. Given two sequences and the unit cost function, we refer to the sequence with less cost as the cheaper one whereas the other with more cost as more expensive. For every marking explored, we keep the Parikh vector of its cheapest previous sequence and neglect the more expensive sequences, which is present in Fig. 4.7a. To clarify, m_i represents the initial marking, and the ellipsis and cloud shape represent part of the state space that we can ignore.

In Fig. 4.7a, firing transition $(\gg, t_j)$ or $(\gg, t_k)$ from marking m_j results in marking m_k. Given the unit cost function, the cost of both transitions $(\gg, t_j)$ and $(\gg, t_k)$ are 1. Thus, m_k has two previous sequences with the same cost, i.e., sequences $\sigma_1 = \langle \ldots, (\gg, t_j) \rangle$ and $\sigma_2 = \langle \ldots, (\gg, t_k) \rangle$. When we determine the feasibility of m_k during cache set checking, it is possible that subtracting Parikh vector of sequence σ_1 from the new initial solution vector leads to infeasible solution, while subtracting the Parikh vector of sequence σ_2 results in feasible solution. Thus, if we only keep track of Parikh vector of sequence σ_1 for m_k, it remains infeasible after examining the cache set. Therefore, m_k should keep the Parikh vectors of both σ_1 and σ_2.

Likewise, in Fig. 4.7b, both $\sigma_3 = \langle \ldots, (\gg, t_n) \rangle$ and $\sigma_4 = \langle \ldots, (\gg, t_k), (t_n, t_n) \rangle$ from the initial marking result in feasible marking m_n. Given the unit cost function, σ_3 and σ_4 have the same cost, thus we keep Parikh vectors of both sequences for m_n.



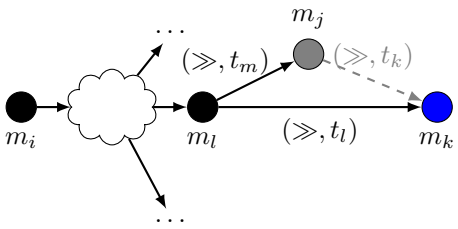
(a) Different sequences of equal cost to m_k

(b) Different sequences of equal cost to m_n

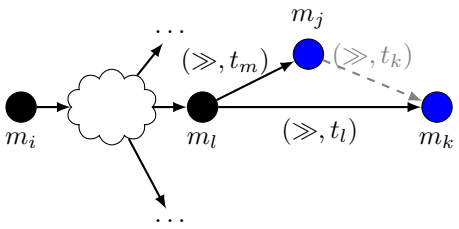
Fig. 4.7: Example of multiple Parikh vectors for a marking

As aforementioned, although a marking explored might possess different previous sequences, we only keep the useful ones, i.e., the Parikh vector of the cheapest sequence. Thus, we abandon Parikh vector of sequences that do not match optimal alignments. During the search, in case a more expensive sequence or a new sequence originating from a loop is found, the algorithm ignores them. We illustrate these two cases in Fig. 4.8 and Fig. 4.9.

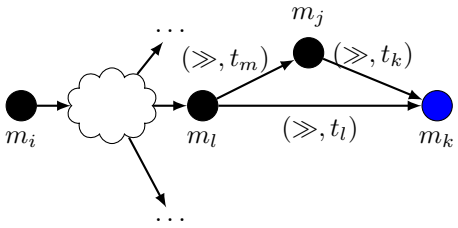
Example in Fig. 4.8 exemplifies the case of discovering a more expensive sequence. In Fig. 4.8a, marking m_k first keep the Parikh vector of sequence $\sigma_5=\langle \ldots, (\gg, t_l)\rangle$. After exploring marking m_j, a more expensive sequence $\sigma_6=\langle \ldots, (\gg, t_j), (\gg, t_k)\rangle$ to m_k is found. Unfortunately, σ_6 never matches optimal alignment. We ignore the corresponding Parikh vector of σ_6 for m_k to prevent useless sequences information.



(a) Reach marking m_k first with sequence $\langle \ldots, (\gg, t_l)\rangle$



(b) Explore marking m_j



(c) Reach marking m_k again with a more expensive sequence $\langle \ldots, (\gg, t_j), (\gg, t_k)\rangle$

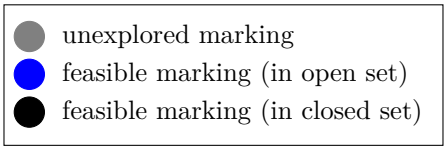


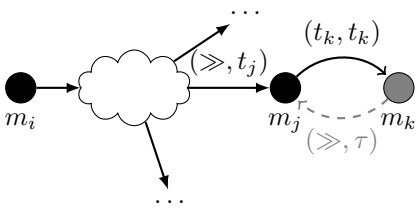
Fig. 4.8: Example of more expensive sequence leading to marking m_k

Example in Fig. 4.9 illustrates the discovery of a sequence from loop. In Fig. 4.9a, marking m_j is first reached with sequence $\sigma_7=\langle \ldots, (\gg, t_j)\rangle$, thus it keeps the corresponding Parikh vector. Then in Fig. 4.9b, firing transition (t_k, t_k) from m_j results in marking m_k, from which the firing of silent transition $(\gg, \tau)$ leads to marking m_j again. Thus, m_j have two previous sequences, i.e., σ_7 and $\sigma_8=\langle \ldots, (\gg, t_j), (t_k, t_k), (\gg, \tau)\rangle$, both of which have the same cost. However, sequence σ_8 results from a loop and contains no useful information. In this case, the Parikh vector of σ_8 is pruned directly.

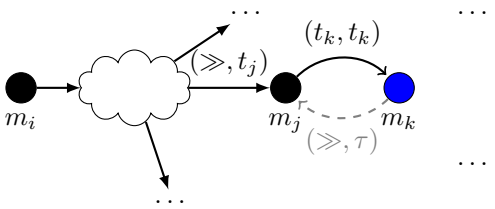
We refer to this approach as naive caching strategy. Its flowchart is shown in Fig. 4.10. The input of the algorithm are the synchronous product net and the cost function. The algorithm runs as follows:

At first, the split point list, open set, cache set and closed set are initialized. Specifically, the open set is implemented as a minimum heap with additional dictionary that stores the index of each marking. Moreover, the open set which priorities the most promising marking during the search. The cache set and closed set are implemented as normal list, which stores infeasible markings and already expanded markings, respectively.

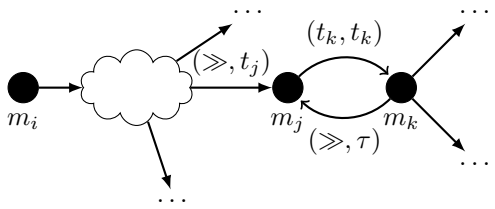
After initialization, the algorithm computes a heuristic for the initial marking and enters the main search loop. As long as the open set is not empty, we iteratively select the most promising marking m from the open set. If m equals the final marking, the search ends and we construct the alignment. Otherwise, we explore all subsequent markings from m.



(a) Reach marking m_k from m_j



(b) Reach marking m_j again from m_k with $(\gg, \tau)$ transition



(c) Reach marking m_j again from m_k with $(\gg, \tau)$ transition

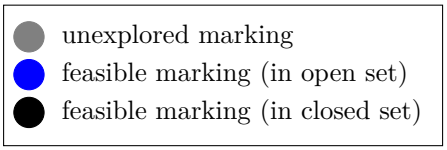


Fig. 4.9: Example of a new sequence from loop to marking m_j

If subsequent marking m' is infeasible, we store it in the cache set. Otherwise, we insert it into the open set. Also, we keep track of the Parikh vector of the sequence for m'. In general, it is possible to explore a marking multiple times with different sequences, which can be less or equally costly than already known sequences. In case the sequence to m' is cheaper than known ones, we keep the corresponding Parikh vector of the cheaper sequence and abandon the Parikh vectors of others. If the sequence is equally costly, we incorporate its corresponding Parikh vector for m'.

When the open set is empty, we first check whether a new split point is found. If no new split point is found, we compute a new solution using regular marking equation for an infeasible marking m'' from the cache set and continue the search. Otherwise, we compute a new heuristic for the initial marking, and check every infeasible marking by subtracting the Parikh vector from the initial solution. After checking, we put all feasible markings into the open set.

Moreover, it is also possible that the introduction of a new split point does not improve the underestimation of the heuristic, i.e., the new initial solution computed is the same as the old one. In this case, all markings in the cache set remain infeasible after checking. To advance the search, we turn to regular A^* search and compute heuristic for infeasible markings before finding a new split point.

In addition to preventing redundant restart during search, another benefit of the caching strategy is the reduced number of markings inserted into open set, since infeasible markings are no longer stored in open set. In the split-point-based algorithm, to guarantee efficient selection of the most promising marking in the search loop, the open set is implemented as a hash-backed minimum heap [4]. However, all markings are kept in the open set regardless the feasibility of their heuristic. This is only advantageous when the search gets stuck and the algorithm turn to regular A^* search, since the algorithm selects the most promising infeasible marking and compute a heuristic for it.

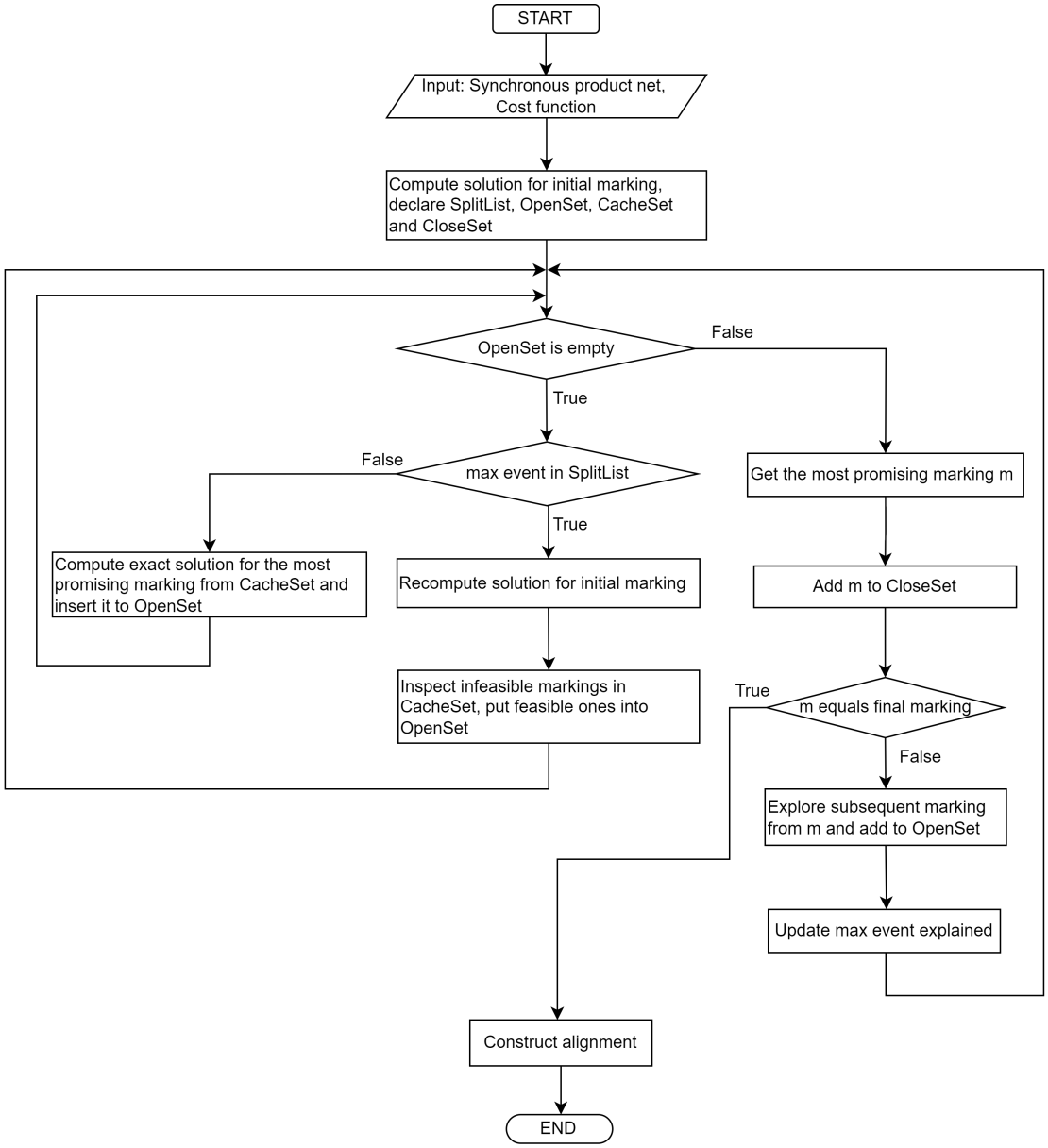
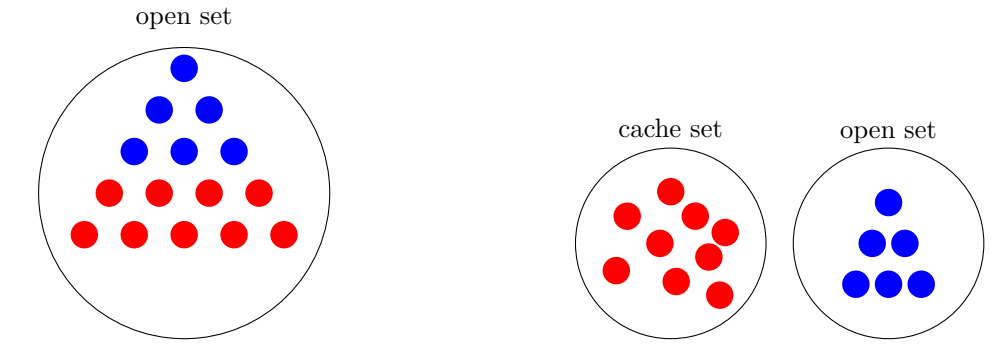


Fig. 4.10: The split-point-based algorithm with naive caching strategy

In contrast, with a cache set to store infeasible markings, only feasible markings are kept in the open set during the search. For example, if there are 16 recently explored markings, 10 of which are infeasible while 6 are feasible, the split-point-based approach keeps them all in the open set and organizes them as a minimum heap. As present in Fig. 4.11a, the 6 blue markings at top are feasible, while the 10 red markings at the bottom are infeasible. The infeasible markings contribute little to the further exploration of the search, as we cannot derive solution for subsequent markings. With caching strategy, we separate feasible markings from infeasible markings as shown in Fig. 4.11b. On the one hand, the cache set does not require a minimum heap to manage infeasible markings. On the other hand, the open set becomes smaller and has fewer heap-related operations.



(a) split-point-based algorithm (b) split-point-based algorithm with caching strategy

Fig. 4.11: The way to store new markings explore before/after using caching strategy

4.4 Modified Caching Strategy

In this section, we propose a sequence propagation method that re-explore markings in the closed set to overcome one side effect of the naive caching strategy as present in Fig. 4.12.

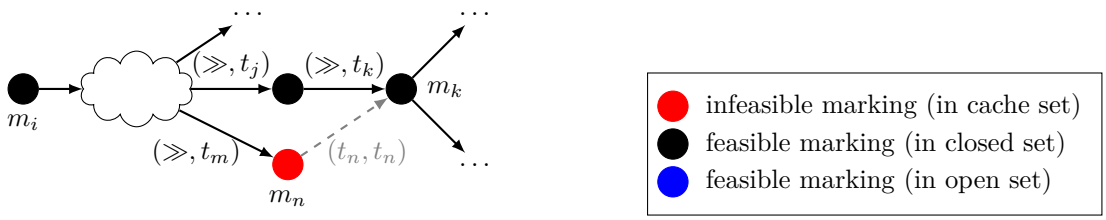
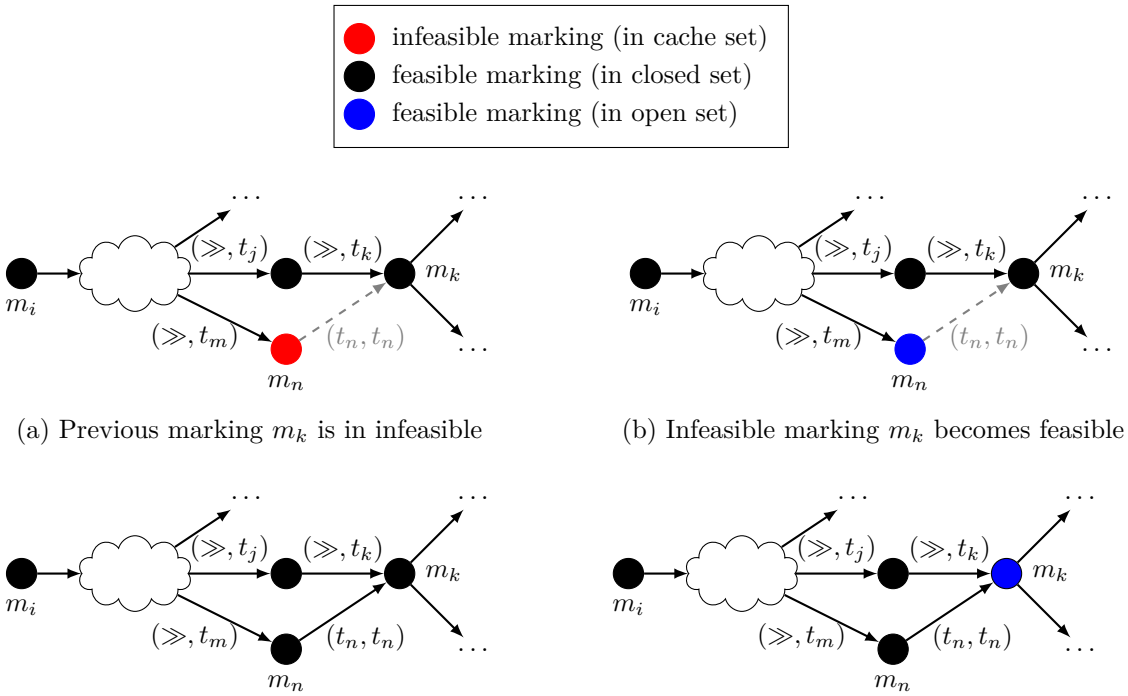


Fig. 4.12: Example of an undiscovered cheaper sequence from marking in cache set to marking in closed set

When multiple sequences exist for an infeasible or feasible marking, we either keep or prune the corresponding Parikh vectors, depending on the cost of the sequence. During the search, if a more expensive sequence is found, we prune it. If a cheaper sequence is found, we keep its Parikh vector and prune the Parikh vectors of other sequences. However, a problem still arises if we keep the markings in closed set after recomputing the initial solution. Certain sequences that normally would be assessed are not considered, thus alternative equally costly or cheaper unexplored sequences that lead to markings in closed set may get ignored. Consequently, the search reaches the final marking with non-optimal alignment.

To illustrate the problem, we exemplify the problem in Fig. 4.13. In Fig. 4.13a, although marking m_k is already in closed set, one of its previous marking m_k is infeasible and stays in the cache set. We assume sequence $\sigma_9 = \langle \ldots, (\gg, t_j), (\gg, t_k) \rangle$ known by m_k is more costly than $\sigma_{10} = \langle \ldots, (\gg, t_m), (t_n, t_n) \rangle$. However, the cheaper sequence $\sigma_1 0$ remains undiscovered as m_n is infeasible and cannot fire transition (t_n, t_n). In Fig. 4.13b, we compute a new heuristic for the initial marking and determine the feasibility of infeasible markings. In this case, m_n becomes feasible after subtracting Parikh vector from the initial solution vector. Subsequently in Fig. 4.13c, the new firing sequence $\sigma_1 0$ is known by m_k. While the original split-point-based approach skip the markings in closed set during search, in our case we need to remove m_k from closed set since a cheaper sequence

is found. Moreover, as sequence σ_{10} is less costly than σ_9, we abandon the Parikh vector of more expensive sequence σ_9 as it never matches optimal alignment. In the following search process, marking m_k will be selected from open set again, and we explore its subsequent markings. Therefore, the updated sequences information, i.e., the new Parikh vector known by marking m_k is propagated recursively to all its subsequent markings.



(a) Previous marking m_k is in infeasible

(b) Infeasible marking m_k becomes feasible

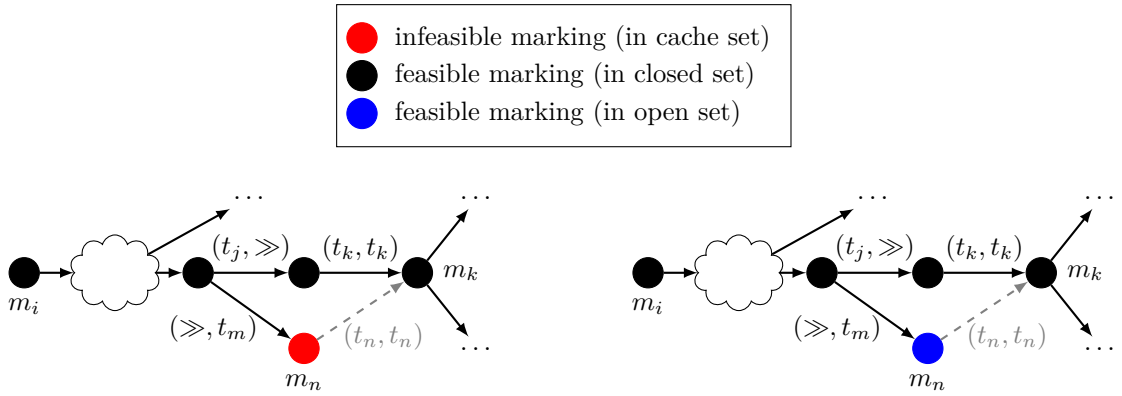
(c) Discover a cheaper sequence $\langle \ldots, (\gg, t_m), (t_n, t_n) \rangle$ to marking m_k

(d) Put marking m_k to open set

Fig. 4.13: Example of removing marking in closed set when cheaper sequence is found

Likewise, as present in Fig. 4.14, marking m_k keeps the Parikh vector of sequence σ_{11}=$\langle \ldots, (t_j, \gg), (t_k, t_k) \rangle$. We assume the unexplored sequence σ_{12}=$\langle \ldots, (\gg, t_m), (t_n, t_n) \rangle$ to m_k is equally costly with the same cost. However, it remains undiscovered due to the infeasiblility of m_n. The sequence σ_{12} to m_k is not found until m_n becomes feasible after cache set checking. In this case, we incorporate the corresponding Parikh vector of σ_{12} for m_k, remove m_k from the closed set and store into the open set. In the following search process, m_k is removed from open set and its subsequent markings are explored again. the newly discovered Parikh vector is propagated to all its subsequent markings.

It is trivial to alter the naive caching strategy in order to apply the additional sequence propagation procedure. When we explore a marking that is already in closed set, we check whether the new sequence leading to it is equally costly or cheaper than already known ones. If not, we remove it from the closed set, incorporate the corresponding Parikh vectors and put it again into open set. In the following search process, its subsequent markings also receive updated information of new Parikh vectors. We refer to this caching strategy with reopening markings from closed set and sequence propagation as modified caching strategy.



(a) Part of the state space when the search gets stuck

(b) Compute new heuristic for infeasible markings m_k in the cache set

(c) Discover equally costly sequence $\langle \ldots, (\gg, t_m), (t_n, t_n) \rangle$ to marking m_k

(d) Put marking m_k to open set

Fig. 4.14: Example of removing marking in closed set when equally costly sequence is found

4.5 Enforce Restart With a Counter

In previous section, we propose to introduce a cache set, which aims to store infeasible markings during search and prevent complete restart when the search gets stuck. However, the strategy goes from one extreme to the other, i.e., while the original split-point-based search restarts completely when stuck, the caching strategy forbids restart as it always reuses previously obtained results. In this section, we propose a trade off implementation that seeks to find a balance between always restart and never restart, which is achieved by setting restart frequency and use a counter to enforce restart.

For instance, by setting the restart frequency to 3, the algorithm turns to complete restart every 3 times when the search gets stuck. Initially, the counter value is set to 0. Every time the search gets stuck and a new initial heuristic is computed, we increase the counter value by 1. When the counter value reaches 3, the algorithm abandon all previously obtained information and restarts completely. Thus, if the search only gets stuck twice before reaching the final marking, the counter is not activated. If the search gets stuck 5 times, the algorithm only restarts from initial marking at the third stuck.

The counter settings that enforce restart is integrated trivially into the algorithm. The more frequent the algorithm restarts, the more homogeneous it is as the split-point-based algorithm. As present in Fig. 4.15, the split-point-based search is equivalent to setting restart frequency to 1, whereas caching strategy is equal to setting restart frequency to infinity. Initially, we set counter value to be 0. Also, we set restart frequency as a threshold

and every time the search gets stuck, we increase the value of counter by 1. If the counter value reaches the threshold, we restart the search completely and abandon all information obtained.

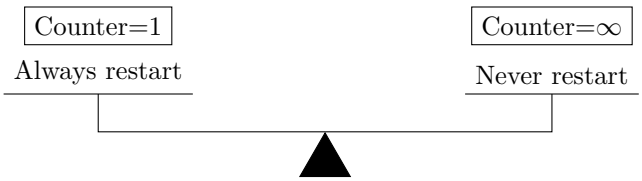


Fig. 4.15: The balance of always restart with and never restart with different counter value

Chapter 5

Evaluation

In this chapter, we present experimental results of the proposed alignment computation algorithms and compare them with existing techniques. At Section 5.1 we first discuss the experimental setup. Then, we compare the experimental results of our approach with existing techniques in Section 5.2. Specifically, we first compare the running time of different techniques, then we analyze the state space traversal efficiency before and after using caching strategy. Finally in Section 5.3, we discuss the threats to the validity of our results.

5.1 Experimental Setup

We have conducted experiments on various real-life event logs. The first one contains events of sepsis cases from a hospital [7]. The second event log is obtained from the financial modules of the ERP system of a regional hospital [6]. The third one is from BPI Challenge 2020 regarding international insurance declaration [35]. The fourth one is from an information system managing road traffic fines [36]. The fifth one is from Conformance Checking Challenge 2019, which records the learning process of medical students installing Central Venous Catheter with ultrasound [8]. The sixth one to the tenth one originate from Conformance Checking Challenge 2020 [9], which contains the execution process of building permit applications. We summarize the features of each log in Table 5.1.

Table 5.1: Information about the event logs

Event Log	**Average trace length**	**Case variants**	**Total events**
Sepsis [7]	14.48	846	15214
Hospital bill [6]	12.51	1020	451359
International declaration [35]	11.18	753	72151
Road traffic fine [36]	3.73	231	561470
CCC2019 [8]	34.85	20	697
CCC2020-A [9]	20.42	774	30493
CCC2020-B [9]	22.62	870	32875
CCC2020-C [9]	20.92	749	28255
CCC2020-D [9]	22.48	723	28440
CCC2020-E [9]	20.15	815	29065

Table 5.2: The number of places and transitions in process models

	model 0.05		**model 0.20**		**model 0.40**		**model 0.80**	
Event log	**place**	**transition**	**place**	**transition**	**place**	**transition**	**place**	**transition**
Sepsis	28	36	23	24	22	22	21	18
Hospital bill	46	60	29	39	20	28	19	25
International declaration	33	50	52	63	32	50	37	44
Road traffic fine	19	23	17	21	20	20	13	16
CCC2019	58	67	48	50	45	47	25	27
CCC2020-A	58	67	48	50	45	47	27	25
CCC2020-B	29	58	60	71	53	56	48	44
CCC2020-C	51	77	48	69	52	61	41	43
CCC2020-D	66	78	57	67	52	58	39	40
CCC2020-E	51	76	52	68	47	54	37	40

Given the event logs, we discover process models with the process mining tool in ProM [5]. Specifically, the inductive miner (infrequent) is used to derive models with noise threshold set to 0.05, 0.20, 0.40, and 0.80. In general, increasing the noise threshold would filter out more infrequent behaviors in the event log, thus the number of places and transition decreases in the mined model. The features of different models, including the number of places and transitions, are listed in Table 5.2.

We run experiment with 5 different algorithms. Specifically, the first one is the regular A^* search [16], and the second one refers to the split-point-based search [4]. They two served as baseline. As for the third one, we applied caching strategy proposed in Section 4.3 for the split-point-based search, which means the search never restarts but checks the cache set and continues the search. As for the fourth one, we conducted an additional sequence propagation step proposed in Section 4.4 on top of caching strategy. To measure the difference between restart search and not restart the search, we further introduced a counter proposed in Section 4.5 to enforce the search to restart completely when getting stuck. For example, if the counter is set to 2, then after every two rounds of search, the algorithm completely re-initializes open set and closed set, i.e., abandons all known information and restart the search from the initial marking. In the remainder of the thesis, we refer to each algorithm with abbreviations listed in Table 5.3.

The proposed algorithms with caching strategy and counter settings have been implemented with the open-source Python library PM4Py [37]. We uploaded related codes, event logs and models to an elastic cloud server, and conducted all experiments in single-threaded mode. The server was equipped with a 3.5 GHz Intel Xeon Platinum 8369HC CPU, 4 GB of RAM and 40 GB of Disk. The running time was not bounded, as long as the out-of-memory error did not occur. During the experiment, we observed no memory overflow. We recorded the program running times of all techniques. To guarantee the credibility of the results, the alignment for any given trace was computed three times and we calculated the average statistics.

Table 5.3: Algorithms used for experiments

Algorithm	**Abbreviation**
regular A^* search	
split-point-based search	spb
split-point-based search & naive caching strategy	ncs
split-point-based search & modified caching strategy	mcs
split-point-based search & modified caching strategy & counter	ct

5.2 Results Analysis

In this section, we present the experiment results and compare the performance of the proposed algorithm with existing alignment techniques.

As for the experiment results, the most important statistics is the running time and the number of non-optimal alignment, which indicates the efficiency and efficacy of the algorithm. To further analyze the bottleneck of each algorithm variation, we record the time to compute the heuristic, as well as the number of regular linear programming and incremental linear programming problems solved. Also, we record the time spent on queue (open set) related operations, along with the number of insertions, updates and removal from open-set. In addition, we record the number of arcs and markings visited to measure the efficiency of state-space traversal.

We expect the regular A^* search in [16] to be the slowest due to the number of linear programming problems it needs to solve. The split-point-based algorithm from [4] is expected to consume the second-longest time, as it is cumbered with redundant restart that abandons all prior knowledge accumulated during the last round of search. Moreover, we expect naive caching strategy to promote the performance of split-point-based approach, while non-optimal alignments occur. Furthermore, It is expected that the modified caching strategy could guarantee optimal alignment and outperform split-point-based approach. In addition, we assume a trade-off between always restarting the search and never restarting exists. Thus, we introduce a counter to enforce the search restarts after certain rounds of search.

Considering the results showing similar patterns, we present the results of Hospital bill and CCC2020-B in Table 5.4 and Table 5.5, and leave the rest results in appendices. For the time column, total refers to the total running time whereas heuristic and queue refer to the time spent on heuristic and open set. As for column state space explored, states and arcs refer to the number of states and arcs traversed during the search. As for queue operations, total refers to the sum of insertions, removal, and update operations. As for the LP number, we refer to the regular linear programming defined in Definition 3.14 as small LP and the incremental one defined in Definition 3.17 as big LP, as the latter is more complex and requires more time to solve. The restart column refers to the number of complete restarts from the initial marking during the search. For each column, we apply a blue-red coloring scheme to visualize the respective magnitude of a particular trend within a column. While blue color represents the value is smaller than average, red color indicates the value is greater than average.

First, we briefly the performance of two baseline algorithms, i.e., the regular A^* search and split-point-based search. For event logs Sepsis, Hospital bill, CCC2019 and CCC2020-B,

Table 5.4: The alignment results for CCC2020-B

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	206.3	139.1	9.5	6.7E+05	3.4E+06	3.8E+06	2.8E+06	9.6E+05	0.0E+00	7.1E+04	0.0	0.0	0
spb	697.9	129.8	408.3	1.3E+06	6.6E+06	4.4E+06	2.5E+06	1.3E+06	6.2E+05	1.3E+04	4227.0	4227.0	0
ncs	350.7	127.0	123.5	5.6E+05	2.8E+06	2.0E+06	8.9E+05	5.7E+05	5.0E+05	7.3E+03	4556.7	0.0	0
mcs	491.4	133.0	187.4	7.3E+05	3.6E+06	2.4E+06	1.2E+06	7.4E+05	4.5E+05	7.0E+03	4694.0	0.0	0
ct1	1101.9	146.9	602.8	1.8E+06	9.0E+06	5.5E+06	2.7E+06	1.8E+06	1.0E+06	1.5E+04	4585.0	4585.0	0
ct2	783.9	146.9	372.5	1.3E+06	6.5E+06	4.1E+06	2.0E+06	1.3E+06	8.7E+05	1.1E+04	4717.3	1966.3	0
ct3	700.9	143.4	319.7	1.1E+06	5.6E+06	3.6E+06	1.7E+06	1.1E+06	7.5E+05	1.0E+04	4711.7	1134.3	0
ct4	621.9	141.7	268.1	9.7E+05	4.9E+06	3.1E+06	1.5E+06	9.8E+05	6.4E+05	9.2E+03	4721.3	629.7	0
ct5	611.8	138.6	264.2	9.5E+05	4.7E+06	3.0E+06	1.5E+06	9.5E+05	6.2E+05	8.9E+03	4682.0	472.3	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	5437.1	5171.0	59.9	8.7E+05	6.7E+06	1.2E+07	7.3E+06	5.0E+06	0.0E+00	2.7E+06	0.0	0.0	0
spb	113.9	33.2	53.7	1.6E+05	1.1E+06	8.8E+05	6.7E+05	1.7E+05	4.6E+04	1.5E+03	1430.3	1430.3	0
ncs	121.4	71.5	24.2	1.1E+05	6.5E+05	3.3E+05	8.6E+04	1.3E+05	1.1E+05	2.3E+04	1441.3	0.0	0
mcs	69.3	32.6	11.7	9.8E+04	6.0E+05	2.3E+05	1.2E+05	9.9E+04	1.2E+04	1.7E+03	1426.0	0.0	0
ct1	108.8	32.7	37.7	1.6E+05	1.1E+06	6.6E+05	4.6E+05	1.7E+05	3.1E+04	1.5E+03	1425.0	1425.0	0
ct2	82.6	32.8	20.1	1.2E+05	7.6E+05	3.9E+05	2.4E+05	1.2E+05	2.5E+04	1.5E+03	1426.0	406.0	0
ct3	77.6	32.4	16.9	1.1E+05	7.1E+05	3.2E+05	1.9E+05	1.1E+05	1.8E+04	1.5E+03	1423.0	225.3	0
ct4	72.8	32.5	14.0	1.0E+05	6.4E+05	2.7E+05	1.5E+05	1.0E+05	1.5E+04	1.5E+03	1428.3	107.7	0
ct5	72.0	33.1	13.0	1.0E+05	6.2E+05	2.5E+05	1.4E+05	1.0E+05	1.4E+04	1.8E+03	1430.3	29.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3342.9	3196.6	31.5	5.3E+05	3.4E+06	7.1E+06	4.1E+06	3.0E+06	0.0E+00	1.8E+06	0.0	0.0	0
spb	117.7	47.8	47.3	1.6E+05	8.6E+05	7.8E+05	5.8E+05	1.7E+05	3.5E+04	1.5E+03	2194.3	2194.3	0
ncs	77.2	49.2	10.7	9.9E+04	4.5E+05	2.4E+05	4.2E+04	1.0E+05	9.4E+04	2.5E+03	2223.0	0.0	0
mcs	77.2	46.5	9.7	9.6E+04	4.8E+05	2.2E+05	1.2E+05	9.8E+04	9.3E+03	1.5E+03	2190.3	0.0	0
ct1	114.1	47.0	35.5	1.6E+05	8.6E+05	6.4E+05	4.5E+05	1.6E+05	2.4E+04	1.5E+03	2195.0	2195.0	0
ct2	92.1	46.9	19.9	1.2E+05	6.3E+05	3.9E+05	2.5E+05	1.3E+05	1.6E+04	1.5E+03	2190.0	849.3	0
ct3	83.8	46.7	14.1	1.1E+05	5.5E+05	2.9E+05	1.7E+05	1.1E+05	1.2E+04	1.5E+03	2186.3	341.7	0
ct4	81.1	47.2	11.8	1.0E+05	5.1E+05	2.6E+05	1.4E+05	1.0E+05	9.8E+03	1.5E+03	2196.3	158.3	0
ct5	78.5	46.7	10.3	9.8E+04	4.9E+05	2.3E+05	1.3E+05	1.0E+05	9.1E+03	1.5E+03	2186.3	51.7	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	872.4	835.2	9.4	3.2E+05	1.1E+06	2.1E+06	1.2E+06	9.4E+05	0.0E+00	5.0E+05	0.0	0.0	0
spb	115.3	79.0	24.6	1.3E+05	4.3E+05	5.4E+05	4.0E+05	1.4E+05	2.6E+03	1.5E+03	3430.3	3430.3	0
ncs	105.6	92.7	4.7	7.3E+04	2.0E+05	2.0E+05	5.8E+04	7.5E+04	6.8E+04	2.4E+03	3668.7	0.0	0
mcs	91.0	78.1	3.2	7.2E+04	2.2E+05	1.6E+05	8.0E+04	7.4E+04	3.0E+03	1.7E+03	3436.0	0.0	0
ct1	115.4	78.0	22.0	1.3E+05	4.3E+05	5.1E+05	3.7E+05	1.4E+05	2.2E+03	1.5E+03	3423.0	3423.0	0
ct2	103.0	79.2	11.2	1.0E+05	3.2E+05	3.1E+05	2.0E+05	1.1E+05	2.8E+03	1.5E+03	3436.0	1344.0	0
ct3	97.6	78.7	7.6	9.0E+04	2.8E+05	2.4E+05	1.5E+05	9.1E+04	2.4E+03	1.5E+03	3435.7	669.0	0
ct4	95.7	78.3	6.5	8.6E+04	2.6E+05	2.2E+05	1.3E+05	8.7E+04	2.8E+03	1.5E+03	3425.7	441.3	0
ct5	93.6	78.5	4.8	7.9E+04	2.4E+05	1.8E+05	1.0E+05	8.0E+04	3.0E+03	1.5E+03	3433.7	206.3	0

Table 5.5: The alignment results for Hospital bill

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	13757.3	12430.4	482.4	9.1E+06	4.9E+07	7.0E+07	4.7E+07	2.4E+07	0.0E+00	8.2E+06	0.0	0.0	0
spb	17779.9	10300.9	5683.5	9.2E+06	5.2E+07	4.7E+07	3.2E+07	1.5E+07	7.2E+05	6.4E+06	19696.3	19696.3	0
ncs	12492.0	8149.9	3136.2	6.4E+06	2.8E+07	3.0E+07	1.6E+07	1.1E+07	3.5E+06	5.2E+06	20125.3	0.0	4
mcs	12993.4	8299.2	3178.5	6.5E+06	2.8E+07	3.1E+07	1.9E+07	1.1E+07	6.3E+05	5.3E+06	20077.3	0.0	0
ct1	17139.0	9890.6	4928.2	9.4E+06	5.2E+07	4.1E+07	2.6E+07	1.5E+07	8.6E+05	6.4E+06	19370.3	19370.3	0
ct2	14587.6	9092.6	3685.5	8.4E+06	3.8E+07	3.6E+07	2.2E+07	1.3E+07	5.9E+05	5.7E+06	22028.3	9321.0	0
ct3	14286.3	9007.0	3532.0	7.8E+06	3.5E+07	3.5E+07	2.2E+07	1.3E+07	5.6E+05	5.6E+06	23918.0	5545.7	0
ct4	13186.5	8431.4	3195.9	7.0E+06	3.0E+07	3.2E+07	2.0E+07	1.1E+07	5.3E+05	5.2E+06	25340.0	4263.0	0
ct5	12949.2	8285.3	3135.4	6.7E+06	2.9E+07	3.2E+07	2.0E+07	1.1E+07	5.2E+05	5.0E+06	25748.3	1891.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3520.6	3076.3	182.6	3.5E+06	2.1E+07	2.8E+07	2.0E+07	7.5E+06	0.0E+00	2.3E+06	0.0	0.0	0
spb	896.8	356.4	332.3	1.7E+06	8.3E+06	7.6E+06	5.4E+06	1.9E+06	3.4E+05	1.7E+05	9997.7	9997.7	0
ncs	688.1	320.2	147.2	1.6E+06	6.7E+06	4.2E+06	4.1E+05	1.7E+06	2.1E+06	1.5E+05	10627.7	0.0	1
mcs	726.4	317.5	148.7	1.6E+06	6.7E+06	4.6E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10470.0	0.0	0
ct1	832.7	337.6	203.5	1.7E+06	8.5E+06	5.7E+06	3.2E+06	1.9E+06	5.8E+05	1.7E+05	10445.3	10445.3	0
ct2	755.0	328.3	160.5	1.7E+06	7.0E+06	4.9E+06	2.6E+06	1.8E+06	4.6E+05	1.5E+05	10620.7	2527.0	0
ct3	748.1	334.5	152.7	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.8E+06	4.5E+05	1.5E+05	10619.0	520.0	0
ct4	735.7	326.7	149.3	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10739.0	181.0	0
ct5	734.0	329.9	148.1	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10711.7	67.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	487.6	426.9	136.6	7.0E+05	2.8E+06	4.2E+06	3.1E+06	1.1E+06	0.0E+00	3.4E+05	0.0	0.0	0
spb	365.3	221.4	70.0	8.0E+05	3.1E+06	3.6E+06	2.6E+06	9.1E+05	6.4E+04	1.0E+05	8039.0	8039.0	0
ncs	297.8	210.1	6.8	7.5E+05	2.6E+06	1.7E+06	1.2E+05	8.5E+05	7.6E+05	1.0E+05	7993.3	0.0	0
mcs	310.6	208.9	7.0	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.6E+04	1.0E+05	7942.3	0.0	0
ct1	322.1	206.9	17.2	7.6E+05	3.0E+06	2.1E+06	1.1E+06	8.7E+05	8.2E+04	1.0E+05	7963.3	7963.3	0
ct2	315.0	210.8	9.2	7.7E+05	2.7E+06	1.9E+06	9.4E+05	8.8E+05	5.9E+04	1.0E+05	7958.7	2128.7	0
ct3	316.5	214.7	7.4	7.6E+05	2.7E+06	1.8E+06	9.0E+05	8.6E+05	5.6E+04	1.0E+05	7943.7	399.3	0
ct4	312.0	211.2	6.9	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.6E+04	1.0E+05	7985.0	81.0	0
ct5	313.1	213.2	6.8	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.5E+04	1.0E+05	7934.7	20.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	341.5	287.6	134.4	7.1E+05	2.5E+06	3.7E+06	2.7E+06	9.6E+05	0.0E+00	2.3E+05	0.0	0.0	0
spb	368.8	226.2	69.7	8.1E+05	2.8E+06	3.6E+06	2.6E+06	9.2E+05	5.0E+04	1.0E+05	9139.3	9139.3	0
ncs	288.6	208.1	5.3	7.5E+05	2.4E+06	1.7E+06	1.1E+05	8.5E+05	7.6E+05	1.0E+05	8340.7	0.0	0
mcs	298.2	205.9	5.5	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8275.3	0.0	0
ct1	306.9	204.6	14.4	7.5E+05	2.7E+06	2.0E+06	1.1E+06	8.5E+05	5.0E+04	1.0E+05	8264.7	8264.7	0
ct2	301.7	208.0	7.2	7.6E+05	2.4E+06	1.8E+06	9.2E+05	8.6E+05	4.8E+04	1.0E+05	8238.7	2084.3	0
ct3	304.0	212.1	5.7	7.5E+05	2.4E+06	1.8E+06	8.8E+05	8.5E+05	4.8E+04	1.0E+05	8295.7	324.7	0
ct4	301.0	209.4	5.4	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8281.0	84.0	0
ct5	300.5	209.6	5.4	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8257.3	12.3	0

C, D, E, when log fitness is close to 1, regular A^* search outperforms split-point-based search. However, when the log fitness decreases, we observe that the regular one performs worse than split-point-based search. This observation matches the conclusions present in [4], and the reason behind is that splitting the trace does not guarantee to find a better solution. Thus the overhead of incrementally adding split points is higher than the regular A^* search.

Then we compare the performance of split-point-based search before and after applying naive caching strategy. On the whole, the total running time reduces mainly due to reduced open set related time. However, we observe that using the naive caching strategy leads to non-optimal alignments. For example, for event log Hospital bill there are 5 traces computed with non-optimal alignments, which consist of 0.5% of traces in the log. The factor that contributes to this situation is that we keep the markings in closed set and neglect certain Parikh vectors which should be assessed. The Parikh vector that matches a cheaper sequence is ignored and the final marking is reached with a non-optimal alignment. Also, in some experiments, we observed that the naive caching strategy performs worse. Take log CCC20-B with model 0.20 as an example, after using caching strategy, the time spent on heuristic computation doubled as a result of an increasing number of linear programming solved. What accounts for the rising number is similar, i.e., some Parikh vectors are not correctly assessed, thus the underlying split-point-based algorithm consistently turns to regular A^* search and explores the entire state space. Another reason is that the ncs method advances the search without restart, thus the new split point added always has a bigger index than already existing split points. In contrast, spb restarts completely and the index of the new split point is not necessarily bigger than all other split point. The different way of adding split point for ncs potentially leads to increasing number of complex linear programming and more time spent on heuristic computation.

With modified caching strategy (mcs), the overall running time rises slightly compared to ncs, while the side effect of non-optimal alignments are solved. One reason is that the modified caching strategy allows reopening markings in closed set, which checks and updates Parikh vectors for reachable markings.

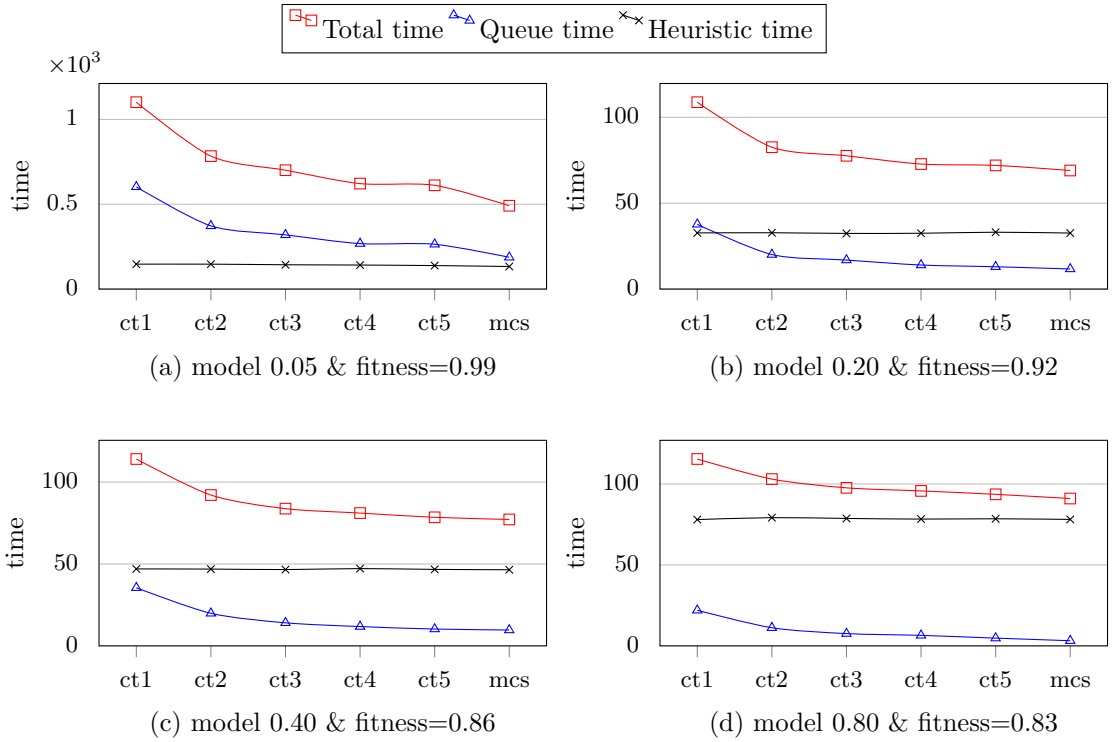
In Table 5.6 we present cases where ncs performs worse than spb (without caching) and mcs. The modified caching strategy outperforms others on log and model where the naive one fails.

Table 5.6: Alignments results in which naive caching performs the worst

Event Log	**Model**	**spb**	**ncs**	**mcs**
CCC2020-A	model 0.80	75.3	89.0	65.4
CCC2020-B	model 0.20	113.9	121.4	69.3
CCC2020-C	model 0.40	92.2	100.7	77.9
CCC2020-D	model 0.40	98.1	117.2	83.0
CCC2020-E	model 0.40	65.5	71.3	59.7

Moreover, there is clear trend when reducing the restart frequency as shown in Fig. 5.1 and Fig. 5.2. To clarify, ct1 indicates that we set restart frequency to 1, ct2 means restart every twice, etc.

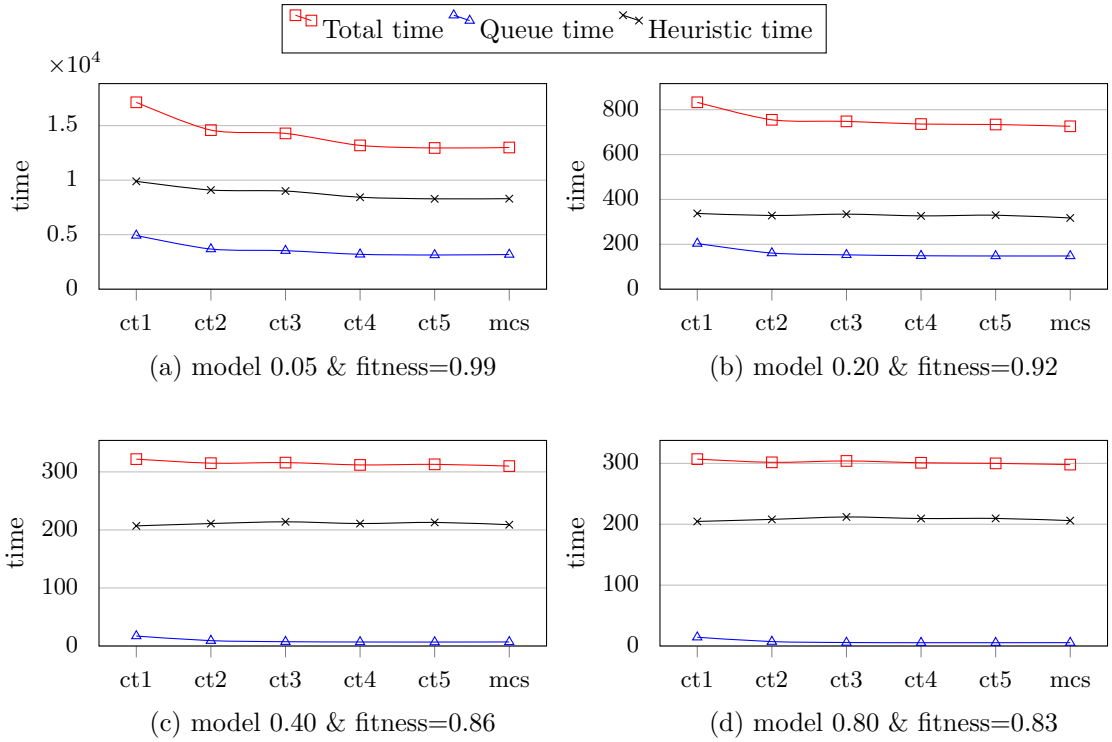
In general, fewer restarts contributes to better running time. When the log fitness is close



(a) model 0.05 & fitness=0.99 (b) model 0.20 & fitness=0.92

(c) model 0.40 & fitness=0.86 (d) model 0.80 & fitness=0.83

Fig. 5.1: The time trend over increasing counter value for CCC2020-B



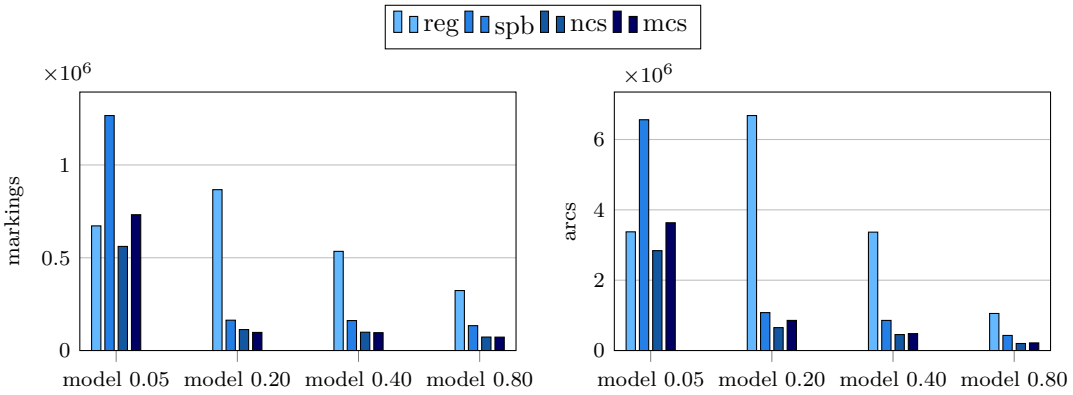
(a) model 0.05 & fitness=0.99 (b) model 0.20 & fitness=0.92

(c) model 0.40 & fitness=0.86 (d) model 0.80 & fitness=0.83

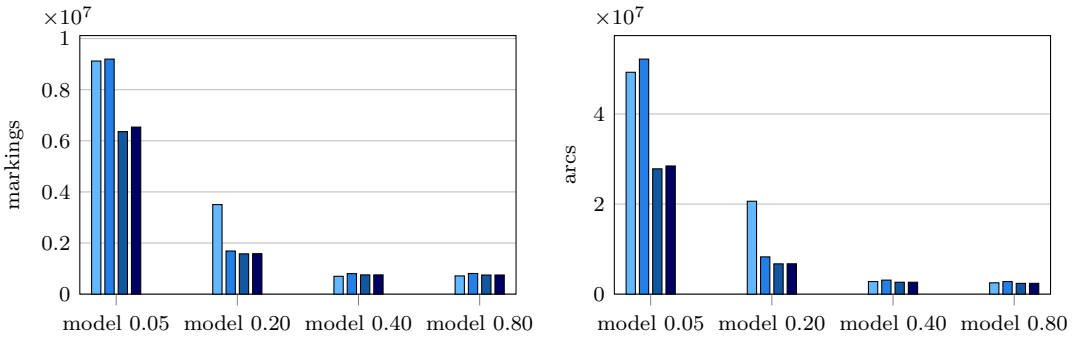
Fig. 5.2: The time trend over increasing counter value for Hospital bill

to 1, i.e., when the log fits perfectly to the model, the drop in the total running time is more significant if we forbid frequent restarts. When the log fits less to the model, the total reduction of the running time is less prominent, as the heuristic computation time becomes the bottleneck, which is not solved by caching strategy. As for heuristic computation time, it remains unchanged or fluctuates little in most cases.

In Fig. 5.3, we compare the state-space traversal efficiency of two baseline approaches with two caching strategies, which is measured by the number of states and arcs traversed during the search. As aforementioned, due to the frequent restart of split-point-based approach (spb) , the state space is repeatedly explored. Thus, in some cases, the number of markings and arcs traversed is the biggest. For example, for both logs with model 0.05, the traversal efficiency for spb is the worst. The undesired repeated exploration is compensated when caching strategy is applied, as we observe a great reduction in terms of markings and arcs traversed. As for models 0.20, 0.40 and 0.80, the spb approach outperforms regular A^* (reg), as it avoids traversing the entire state space. Also, the application of both caching strategies further promotes performance. Compared to the modified caching strategy (mcs), the naive one (ncs) traverses state space slightly more efficiently.



(a) The markings and arcs traversed for CCC2020-B



(b) The markings and arcs traversed for Hospital bill

Fig. 5.3: The number of markings and arcs traversed over different models

5.3 Threats to Validity

In this section, we discuss a potential problem that could jeopardize the insights gained from the results.

The method considers the computation from a pure control-flow perspective, thus the model quality could greatly affect the performance. We applied Inductive Miner (infrequent) to discover models from event logs, which leaves out infrequent behaviors from logs. Thus, the model discovered could be incapable of describing real/intended process and is inappropriate to be used reference model.

Chapter 6

Discussion

In this section, we discuss essential observations in experiment results and then seek to explain the reasons behind.

1. Introducing naive or modified caching strategy for split-point-based algorithm promotes overall performance in terms of running time and state space traversal efficiency.
2. While naive caching strategy approximates optimal alignments, modified caching strategy guarantees optimal alignments.
3. The trade-off between always and never restarting the search does not exist, i.e., the split-point-based algorithm benefits from reusing previously obtained results and not restarting from scratch.

Although split-point-based approach outperforms regular A^* in most cases, it is slower than regular A^* when the model is relatively small or the trace is short. The reason behind is that regular A^* computes normal linear programming and searches the state space quick enough to rule out the need for computing a more complex and time-consuming heuristic.

The disadvantage of split-point-based is compensated by an additional cache set. On the one hand, with a cache set, we no longer need to maintain infeasible markings in the open set, thus greatly reducing insertions and update operations for the open set. On the other hand, the search avoids exploring states repeatedly as it only continues from the frontier of the last round of search. As present in Table 5.4 and Table 5.5, the number of open set-related operations, as well as the number of markings and arcs, traversed get reduced greatly. On the whole, using caching strategy promotes the performance of split-point-based approach.

Although the naive caching strategy is the quickest computation technique in most cases, it leads to a minor proportion of non-optimal alignments. The reason behind is that some sequences that should be assessed are ignored if we never restart the search, the search deviates to an undesired direction before reaching the final marking. Also, this could explain that in some cases a cache set does not contribute to better running time, as the search falls into a wrong direction in state space and introduce more split points when getting stuck, which in turn requires more incremental linear programming to be

solved. Therefore, we classify the naive caching strategy as an alignment approximation technique.

The problem of the aforementioned non-optimal alignments is solved by additional sequence propagation in modified caching strategy. During the search, we evaluate markings in the closed set and check whether there exist unexplored cheaper sequences. While this procedure consumes more time as we remove markings from the closed set, it guarantees optimal alignments and is still quicker than the original split-point-based approach.

Moreover, the results of counter settings imply that setting restart frequency is not beneficial for the search. With different restart frequency, the total running time shows no obvious trade-off between always reusing previously obtained information and always abandon the information. The reason behind is that the split-point-based algorithm benefits from fewer repeated explorations of state space, and infrequent restart indicates fewer repetition.

Chapter 7

Conclusion

In this chapter, we first summarize the main contributions of this thesis, then we present interesting directions for future work.

We focus on alignments computation in this thesis, and the key problem we solved is the repeated exploration of state space for a novel split-point-based search algorithm. We implemented several approaches, which aim to traverse the state space more efficiently by reusing previously obtained results. Unlike regular A^* or split-point-based search that store all markings in the open set regardless of the feasibility of the markings, the proposed caching strategy manages infeasible markings separately in a cache set. Later when the search gets stuck, the algorithm checks the feasibility of markings in the cache set and continues the search from the boundary of the stuck point. Thus, it reduces the total number of markings processed compared to the split-point-based algorithm. Moreover, we introduced additional counter settings to search for the trade-off between always and never restarts the search.

We conducted experiments with real-life event logs and evaluated the results with existing techniques. In general, the proposed caching strategy aids the split-point-based algorithm to traverse state space more efficiently. The efficiency of the proposed caching strategy is proved by counter settings, which enforce restart given a certain preset restart frequency. With increasing counter value, the algorithm computes alignments faster, thus always reusing previously obtained results benefit the search. With promising experiment results, it is shown that the caching strategy promotes the time performance and traversal efficiency of the original split-point-based approach.

7.1 Future Work

We consider the focus of future research to further extend and improve the split-point-based algorithm, which leads to three promising directions. We first consider dynamically reducing the cache set size. During experiments, some infeasible markings never contribute to the search as they stay in the cache set from beginning to end, thus it is possible to trim them after certain rounds of checking by assigning an aging factor to each infeasible marking. Another direction is bidirectional search, which does not necessarily guarantee optimality but potentially reduces the number of markings and arcs traversed. Furthermore, we seek to generalize a way to predict which alignments computation algorithm is

the fastest for a given trace and a model, so that we can select a proper one to use.

Bibliography

[1] Wil M. P. van der Aalst. Process mining: Overview and opportunities. *ACM Trans. Manag. Inf. Syst.*, 3(2):7:1–7:17, 2012. doi: 10.1145/2229156.2229157. URL https://doi.org/10.1145/2229156.2229157.

[2] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition.* Springer, 2016. ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4. URL https://doi.org/10.1007/978-3-662-49851-4.

[3] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models.* Springer, 2018. ISBN 978-3-319-99413-0. doi: 10.1007/978-3-319-99414-7.

[4] Boudewijn F. van Dongen. Efficiently computing alignments - using the extended marking equation. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, volume 11080 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2018. doi: 10.1007/978-3-319-98648-7_12. URL https://doi.org/10.1007/978-3-319-98648-7_12.

[5] Eric Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Prom 6: The process mining toolkit. In Marcello La Rosa, editor, *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010*, volume 615 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL http://ceur-ws.org/Vol-615/paper13.pdf.

[6] Mannhardt Felix. Hospital billing - event log, August 2017. URL https://data.4tu.nl/articles/dataset/Hospital_Billing_-_Event_Log/12705113/1.

[7] Mannhardt Felix. Sepsis cases - event log, December 2016. URL https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1.

[8] Munoz-Gama Jorge, de la Fuente R., Sepúlveda M., and Fuentes R. Conformance checking challenge 2019 (ccc19), February 2019. URL https://data.4tu.nl/articles/dataset/Conformance_Checking_Challenge_2019_CCC19_/12714932/1.

[9] Joos C. A. M. Buijs. Environmental permit application process ('wabo'), May 2014. URL https://data.4tu.nl/collections/Environmental_permit_application_process_WABO_CoSeLoG_project/5065529/1.

[10] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based

on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008. doi: 10.1016/j.is.2007.07.001. URL https://doi.org/10.1016/j.is.2007.07.001.

[11] Alessandro Berti and Wil M. P. van der Aalst. Reviving token-based replay: Increasing speed while improving diagnostics. In Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2019 Satellite event of the conferences: 40th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2019 and 19th International Conference on Application of Concurrency to System Design ACSD 2019, ATAED@Petri Nets/ACSD 2019, Aachen, Germany, June 25, 2019*, volume 2371 of *CEUR Workshop Proceedings*, pages 87–103. CEUR-WS.org, 2019. URL http://ceur-ws.org/Vol-2371/ATAED2019-87-103.pdf.

[12] Sebastian Dunzer, Matthias Stierle, Martin Matzner, and Stephan Baier. Conformance checking: A state-of-the-art literature review. *CoRR*, abs/2007.10903, 2020. URL https://arxiv.org/abs/2007.10903.

[13] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace alignment in process mining: Opportunities for process diagnostics. In Richard Hull, Jan Mendling, and Stefan Tai, editors, *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, volume 6336 of *Lecture Notes in Computer Science*, pages 227–242. Springer, 2010. doi: 10.1007/978-3-642-15618-2_17.

[14] Arya Adriansyah. Aligning observed and modeled behavior (ph. d. thesis). *Eindhoven University of Technology*, 2014.

[15] Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. Tuning alignment computation: An experimental evaluation. In Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2017 Satellite event of the conferences: 38th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2017 and 17th International Conference on Application of Concurrency to System Design ACSD 2017, Zaragoza, Spain, June 26-27, 2017*, volume 1847 of *CEUR Workshop Proceedings*, pages 6–20. CEUR-WS.org, 2017. URL http://ceur-ws.org/Vol-1847/paper01.pdf.

[16] Sebastiaan J. van Zelst, Alfredo Bolt, and Boudewijn F. van Dongen. Computing alignments of event data and process models. *Trans. Petri Nets Other Model. Concurr.*, 13:1–26, 2018. doi: 10.1007/978-3-662-58381-4_1. URL https://doi.org/10.1007/978-3-662-58381-4_1.

[17] Alifah Syamsiyah and Boudewijn F. van Dongen. Improving alignment computation using model-based preprocessing. In *International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019*, pages 73–80. IEEE, 2019. doi: 10.1109/ICPM.2019.00021. URL https://doi.org/10.1109/ICPM.2019.00021.

[18] Andrea Burattin and Josep Carmona. A framework for online conformance checking. In Ernest Teniente and Matthias Weidlich, editors, *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, volume 308 of *Lecture Notes in Business Information Pro-*

cessing, pages 165–177. Springer, 2017. doi: 10.1007/978-3-319-74030-0_12. URL https://doi.org/10.1007/978-3-319-74030-0_12.

[19] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *Int. J. Data Sci. Anal.*, 8(3):269–284, 2019. doi: 10.1007/s41060-017-0078-6.

[20] Daniel Schuster and Sebastiaan J. van Zelst. Online process monitoring using incremental state-space expansion: An exact algorithm. In Dirk Fahland, Chiara Ghidini, Jörg Becker, and Marlon Dumas, editors, *Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings*, volume 12168 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2020. doi: 10.1007/978-3-030-58666-9_9. URL https://doi.org/10.1007/978-3-030-58666-9_9.

[21] Rashid Zaman, Marwan Hassani, and Boudewijn F. van Dongen. A framework for efficient memory utilization in online conformance checking. *CoRR*, abs/2112.13640, 2021. URL https://arxiv.org/abs/2112.13640.

[22] Massimiliano de Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2013. doi: 10.1007/978-3-642-40176-3_10.

[23] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98 (4):407–437, 2016. doi: 10.1007/s00607-015-0441-1. URL https://doi.org/10.1007/s00607-015-0441-1.

[24] Farbod Taymouri and Josep Carmona. A recursive paradigm for aligning observed behavior of large structured process models. In *International Conference on Business Process Management*, pages 197–214. Springer, 2016.

[25] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Single-entry single-exit decomposed conformance checking. *Inf. Syst.*, 46:102–122, 2014. doi: 10.1016/j.is.2014.04.003.

[26] Martin Bauer, Han van der Aa, and Matthias Weidlich. Estimating process conformance by trace sampling and result approximation. *EMISA Forum*, 40(1):17–18, 2020.

[27] Mohammadreza Fani Sani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. Conformance checking approximation using subset selection and edit distance. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, volume 12127 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2020. doi: 10.1007/978-3-030-49435-3_15.

[28] Lluís Padró and Josep Carmona. Approximate computation of alignments of business processes through relaxation labelling. In Thomas T. Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling, editors, *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, volume 11675 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2019. doi: 10.1007/978-3-030-26619-6_17.

[29] Vincent Bloemen, Sebastiaan J. van Zelst, Wil M. P. van der Aalst, Boudewijn F. van Dongen, and Jaco van de Pol. Aligning observed and modelled behaviour by maximizing synchronous moves and using milestones. *Inf. Syst.*, 103:101456, 2022. doi: 10.1016/j.is.2019.101456.

[30] Dexter Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997. ISBN 978-0-387-94907-9.

[31] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Memory-efficient alignment of observed and modeled behavior. *BPM Center Report*, 3:1–44, 2013.

[32] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi: 10.1007/BF01386390. URL `https://doi.org/10.1007/BF01386390`.

[33] Judea Pearl. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984. ISBN 978-0-201-05594-8.

[34] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[35] Boudewijn F. van Dongen. BPI Challenge 2020: International Declarations. 3 2020. doi: 10.4121/uuid:2bbf8f6a-fc50-48eb-aa9e-c4ea5ef7e8c5. URL `https://data.4tu.nl/articles/dataset/BPI_Challenge_2020_International_Declarations/12687374`.

[36] Massimiliano de Leoni and Mannhardt Felix. Road traffic fine management process, February 2015. URL `https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1`.

[37] Alessandro Berti, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. Process mining for python (pm4py): Bridging the gap between process- and data science. *CoRR*, abs/1905.06169, 2019.

Appendices

Table 1: The alignment results for Sepsis

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	Time	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	132.8	87.0	5.1	2.8E+05	2.0E+06	2.1E+06	1.6E+06	5.1E+05	0.0E+00	5.5E+04	0.0	0.0	0
spb	146.6	15.3	94.5	1.7E+05	1.3E+06	9.4E+05	6.6E+05	1.8E+05	1.0E+05	4.7E+03	783.7	783.7	0
ncs	90.6	14.8	36.1	1.4E+05	1.0E+06	5.0E+05	2.0E+05	1.4E+05	1.5E+05	5.6E+03	637.7	0.0	0
mcs	102.5	15.7	37.8	1.4E+05	1.0E+06	5.5E+05	3.5E+05	1.5E+05	5.7E+04	6.1E+03	653.7	0.0	0
ct1	117.7	14.3	50.4	1.7E+05	1.3E+06	6.2E+05	3.7E+05	1.7E+05	7.6E+04	3.5E+03	768.0	768.0	0
ct2	99.1	14.0	37.8	1.4E+05	1.0E+06	5.5E+05	3.5E+05	1.4E+05	5.8E+04	4.9E+03	635.7	87.0	0
ct3	103.2	15.7	37.9	1.4E+05	1.1E+06	5.5E+05	3.5E+05	1.5E+05	5.8E+04	6.1E+03	640.0	13.7	0
ct4	102.7	16.1	37.2	1.4E+05	1.0E+06	5.5E+05	3.5E+05	1.5E+05	5.7E+04	6.2E+03	655.0	2.7	0
ct5	98.8	15.2	36.4	1.4E+05	1.0E+06	5.4E+05	3.4E+05	1.4E+05	5.5E+04	5.8E+03	635.0	0.0	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	43.2	37.7	2.1	4.7E+04	2.4E+05	3.2E+05	2.4E+05	8.6E+04	0.0E+00	2.5E+04	0.0	0.0	0
spb	22.1	9.9	8.2	3.2E+04	1.5E+05	1.6E+05	1.1E+05	3.5E+04	1.7E+04	3.3E+03	444.7	444.7	0
ncs	17.9	9.1	3.7	3.1E+04	1.4E+05	9.0E+04	1.4E+04	3.4E+04	4.3E+04	2.9E+03	426.0	0.0	0
mcs	18.7	9.2	3.7	3.1E+04	1.4E+05	1.0E+05	5.5E+04	3.4E+04	1.2E+04	2.9E+03	426.7	0.0	0
ct1	20.3	10.0	4.4	3.4E+04	1.5E+05	1.1E+05	6.4E+04	3.6E+04	1.3E+04	3.1E+03	446.0	446.0	0
ct2	19.8	9.9	4.0	3.2E+04	1.4E+05	1.0E+05	5.7E+04	3.5E+04	1.2E+04	3.0E+03	443.0	132.7	0
ct3	19.2	9.5	3.8	3.2E+04	1.4E+05	1.0E+05	5.6E+04	3.5E+04	1.2E+04	3.0E+03	429.3	57.3	0
ct4	19.0	9.3	3.8	3.2E+04	1.4E+05	1.0E+05	5.5E+04	3.4E+04	1.2E+04	3.0E+03	423.3	21.3	0
ct5	18.8	9.3	3.7	3.1E+04	1.4E+05	1.0E+05	5.5E+04	3.4E+04	1.2E+04	2.9E+03	424.0	1.3	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	52.4	49.6	1.9	3.8E+04	1.0E+05	2.1E+05	1.3E+05	7.9E+04	0.0E+00	3.7E+04	0.0	0.0	0
spb	34.2	25.4	5.3	5.2E+04	1.5E+05	1.8E+05	1.3E+05	5.5E+04	3.2E+03	1.1E+03	1556.0	1556.0	0
ncs	28.8	25.4	0.6	3.6E+04	7.0E+04	7.7E+04	4.9E+03	3.7E+04	3.5E+04	1.2E+03	1555.0	0.0	0
mcs	29.4	25.6	0.6	3.6E+04	7.1E+04	7.8E+04	3.9E+04	3.7E+04	8.8E+02	1.2E+03	1557.0	0.0	0
ct1	34.6	25.5	4.3	6.1E+04	1.2E+05	1.7E+05	1.0E+05	6.3E+04	1.5E+03	1.1E+03	1545.0	1545.0	0
ct2	32.3	26.3	2.1	4.7E+04	9.0E+04	1.1E+05	6.3E+04	4.9E+04	1.1E+03	1.1E+03	1549.0	471.0	0
ct3	31.1	26.2	1.3	4.2E+04	8.0E+04	9.4E+04	4.9E+04	4.3E+04	9.5E+02	1.1E+03	1546.3	153.0	0
ct4	30.5	26.1	1.0	4.0E+04	7.6E+04	8.7E+04	4.5E+04	4.1E+04	8.8E+02	1.2E+03	1563.3	76.7	0
ct5	30.0	25.9	0.9	3.8E+04	7.3E+04	8.2E+04	4.2E+04	3.9E+04	9.0E+02	1.1E+03	1540.7	39.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	86.4	81.7	2.4	4.6E+04	1.6E+05	3.3E+05	2.1E+05	1.2E+05	0.0E+00	6.8E+04	0.0	0.0	0
spb	28.4	16.3	8.1	4.8E+04	1.6E+05	1.9E+05	1.4E+05	5.1E+04	1.0E+03	1.1E+03	1209.7	1209.7	0
ncs	20.0	15.9	0.9	3.3E+04	7.9E+04	7.4E+04	7.9E+03	3.5E+04	3.1E+04	1.3E+03	1194.7	0.0	0
mcs	21.6	16.9	0.9	3.5E+04	8.0E+04	7.5E+04	3.8E+04	3.6E+04	1.6E+03	1.2E+03	1211.0	0.0	0
ct1	27.5	16.5	5.5	5.6E+04	1.3E+05	1.6E+05	1.0E+05	5.8E+04	4.2E+02	1.1E+03	1202.7	1202.7	0
ct2	24.7	17.6	2.6	4.6E+04	1.0E+05	1.1E+05	5.9E+04	4.7E+04	1.6E+03	1.1E+03	1216.3	318.7	0
ct3	22.2	16.6	1.5	3.9E+04	8.7E+04	8.7E+04	4.5E+04	4.1E+04	1.8E+03	1.1E+03	1196.3	77.7	0
ct4	22.2	16.9	1.2	3.8E+04	8.4E+04	8.3E+04	4.2E+04	3.9E+04	2.0E+03	1.1E+03	1218.7	33.3	0
ct5	22.9	17.3	1.2	3.8E+04	8.4E+04	8.2E+04	4.0E+04	3.9E+04	2.7E+03	1.2E+03	1200.0	15.7	0

Table 2: The alignment results for Hospital bill

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	13757.3	12430.4	482.4	9.1E+06	4.9E+07	7.0E+07	4.7E+07	2.4E+07	0.0E+00	8.2E+06	0.0	0.0	0
spb	17779.9	10300.9	5683.5	9.2E+06	5.2E+07	4.7E+07	3.2E+07	1.5E+07	7.2E+05	6.4E+06	19696.3	19696.3	0
ncs	12492.0	8149.9	3136.2	6.4E+06	2.8E+07	3.0E+07	1.6E+07	1.1E+07	3.5E+06	5.2E+06	20125.3	0.0	4
mcs	12993.4	8299.2	3178.5	6.5E+06	2.8E+07	3.1E+07	1.9E+07	1.1E+07	6.3E+05	5.3E+06	20077.3	0.0	0
ct1	17139.0	9890.6	4928.2	9.4E+06	5.2E+07	4.1E+07	2.6E+07	1.5E+07	8.6E+05	6.4E+06	19370.3	19370.3	0
ct2	14587.6	9092.6	3685.5	8.4E+06	3.8E+07	3.6E+07	2.2E+07	1.3E+07	5.9E+05	5.7E+06	22028.3	9321.0	0
ct3	14286.3	9007.0	3532.0	7.8E+06	3.5E+07	3.5E+07	2.2E+07	1.3E+07	5.6E+05	5.6E+06	23918.0	5545.7	0
ct4	13186.5	8431.4	3195.9	7.0E+06	3.0E+07	3.2E+07	2.0E+07	1.1E+07	5.3E+05	5.2E+06	25340.0	4263.0	0
ct5	12949.2	8285.3	3135.4	6.7E+06	2.9E+07	3.2E+07	2.0E+07	1.1E+07	5.2E+05	5.0E+06	25748.3	1891.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3520.6	3076.3	182.6	3.5E+06	2.1E+07	2.8E+07	2.0E+07	7.5E+06	0.0E+00	2.3E+06	0.0	0.0	0
spb	896.8	356.4	332.3	1.7E+06	8.3E+06	7.6E+06	5.4E+06	1.9E+06	3.4E+05	1.7E+05	9997.7	9997.7	0
ncs	688.1	320.2	147.2	1.6E+06	6.7E+06	4.2E+06	4.1E+05	1.7E+06	2.1E+06	1.5E+05	10627.7	0.0	1
mcs	726.4	317.5	148.7	1.6E+06	6.7E+06	4.6E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10470.0	0.0	0
ct1	832.7	337.6	203.5	1.7E+06	8.5E+06	5.7E+06	3.2E+06	1.9E+06	5.8E+05	1.7E+05	10445.3	10445.3	0
ct2	755.0	328.3	160.5	1.7E+06	7.0E+06	4.9E+06	2.6E+06	1.8E+06	4.6E+05	1.5E+05	10620.7	2527.0	0
ct3	748.1	334.5	152.7	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.8E+06	4.5E+05	1.5E+05	10619.0	520.0	0
ct4	735.7	326.7	149.3	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10739.0	181.0	0
ct5	734.0	329.9	148.1	1.6E+06	6.8E+06	4.7E+06	2.5E+06	1.7E+06	4.5E+05	1.5E+05	10711.7	67.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	487.6	426.9	136.6	7.0E+05	2.8E+06	4.2E+06	3.1E+06	1.1E+06	0.0E+00	3.4E+05	0.0	0.0	0
spb	365.3	221.4	70.0	8.0E+05	3.1E+06	3.6E+06	2.6E+06	9.1E+05	6.4E+04	1.0E+05	8039.0	8039.0	0
ncs	297.8	210.1	6.8	7.5E+05	2.6E+06	1.7E+06	1.2E+05	8.5E+05	7.6E+05	1.0E+05	7993.3	0.0	0
mcs	310.6	208.9	7.0	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.6E+04	1.0E+05	7942.3	0.0	0
ct1	322.1	206.9	17.2	7.6E+05	3.0E+06	2.1E+06	1.1E+06	8.7E+05	8.2E+04	1.0E+05	7963.3	7963.3	0
ct2	315.0	210.8	9.2	7.7E+05	2.7E+06	1.9E+06	9.4E+05	8.8E+05	5.9E+04	1.0E+05	7958.7	2128.7	0
ct3	316.5	214.7	7.4	7.6E+05	2.7E+06	1.8E+06	9.0E+05	8.6E+05	5.6E+04	1.0E+05	7943.7	399.3	0
ct4	312.0	211.2	6.9	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.6E+04	1.0E+05	7985.0	81.0	0
ct5	313.1	213.2	6.8	7.5E+05	2.6E+06	1.8E+06	8.8E+05	8.5E+05	5.5E+04	1.0E+05	7934.7	20.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	341.5	287.6	134.4	7.1E+05	2.5E+06	3.7E+06	2.7E+06	9.6E+05	0.0E+00	2.3E+05	0.0	0.0	0
spb	368.8	226.2	69.7	8.1E+05	2.8E+06	3.6E+06	2.6E+06	9.2E+05	5.0E+04	1.0E+05	9139.3	9139.3	0
ncs	288.6	208.1	5.3	7.5E+05	2.4E+06	1.7E+06	1.1E+05	8.5E+05	7.6E+05	1.0E+05	8340.7	0.0	0
mcs	298.2	205.9	5.5	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8275.3	0.0	0
ct1	306.9	204.6	14.4	7.5E+05	2.7E+06	2.0E+06	1.1E+06	8.5E+05	5.0E+04	1.0E+05	8264.7	8264.7	0
ct2	301.7	208.0	7.2	7.6E+05	2.4E+06	1.8E+06	9.2E+05	8.6E+05	4.8E+04	1.0E+05	8238.7	2084.3	0
ct3	304.0	212.1	5.7	7.5E+05	2.4E+06	1.8E+06	8.8E+05	8.5E+05	4.8E+04	1.0E+05	8295.7	324.7	0
ct4	301.0	209.4	5.4	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8281.0	84.0	0
ct5	300.5	209.6	5.4	7.5E+05	2.4E+06	1.8E+06	8.7E+05	8.5E+05	4.8E+04	1.0E+05	8257.3	12.3	0

Table 3: The alignment results for International declarations

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	163.1	135.9	13.1	2.8E+05	1.4E+06	1.8E+06	1.4E+06	4.7E+05	0.0E+00	9.2E+04	0.0	0.0	0
spb	71.1	23.1	30.5	1.6E+05	7.6E+05	8.1E+05	6.1E+05	1.7E+05	3.6E+04	6.6E+03	1000.7	1000.7	0
ncs	53.4	23.1	9.4	1.5E+05	6.8E+05	3.7E+05	1.8E+04	1.6E+05	1.9E+05	6.9E+03	973.0	0.0	0
mcs	55.9	22.8	9.4	1.5E+05	6.8E+05	3.8E+05	2.0E+05	1.6E+05	2.7E+04	6.5E+03	972.7	0.0	0
ct1	61.3	23.2	13.3	1.6E+05	7.3E+05	4.7E+05	2.7E+05	1.7E+05	3.0E+04	6.5E+03	996.7	996.7	0
ct2	58.1	23.1	10.6	1.6E+05	7.1E+05	4.2E+05	2.3E+05	1.7E+05	2.8E+04	6.5E+03	979.3	334.7	0
ct3	55.7	22.8	9.4	1.5E+05	6.9E+05	3.9E+05	2.0E+05	1.6E+05	2.7E+04	6.5E+03	970.7	29.3	0
ct4	55.9	23.0	9.3	1.5E+05	6.9E+05	3.8E+05	2.0E+05	1.6E+05	2.7E+04	6.5E+03	972.7	10.7	0
ct5	56.1	23.3	9.2	1.5E+05	6.9E+05	3.8E+05	2.0E+05	1.6E+05	2.7E+04	6.5E+03	970.7	1.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	6020.7	5559.3	88.9	1.9E+06	1.4E+07	2.2E+07	1.2E+07	9.4E+06	0.0E+00	3.6E+06	0.0	0.0	0
spb	468.2	76.8	246.7	8.5E+05	5.6E+06	3.3E+06	2.2E+06	8.6E+05	3.1E+05	6.4E+03	4994.0	4994.0	0
ncs	300.2	111.8	86.1	6.3E+05	2.9E+06	1.6E+06	4.3E+05	6.5E+05	5.1E+05	2.9E+04	4938.0	0.0	0
mcs	312.3	86.0	75.0	6.1E+05	3.9E+06	1.4E+06	7.2E+05	6.2E+05	1.0E+05	1.1E+04	4975.3	0.0	0
ct1	388.2	79.5	144.6	8.7E+05	4.3E+06	2.5E+06	1.4E+06	8.8E+05	1.6E+05	6.4E+03	4970.3	4970.3	0
ct2	303.8	79.5	92.4	7.1E+05	3.3E+06	1.8E+06	9.4E+05	7.2E+05	1.3E+05	6.5E+03	4981.7	1666.0	0
ct3	305.1	80.7	74.5	6.2E+05	4.0E+06	1.5E+06	7.5E+05	6.3E+05	1.1E+05	7.2E+03	4980.7	270.7	0
ct4	310.6	87.8	73.8	6.1E+05	3.9E+06	1.5E+06	7.3E+05	6.2E+05	1.1E+05	1.2E+04	4973.3	66.0	0
ct5	316.5	89.8	75.0	6.1E+05	4.0E+06	1.5E+06	7.3E+05	6.2E+05	1.0E+05	1.2E+04	4974.7	3.7	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	6740.8	6101.0	103.2	2.4E+06	2.1E+07	2.7E+07	1.8E+07	9.7E+06	0.0E+00	4.1E+06	0.0	0.0	0
spb	3113.2	1195.3	1294.1	2.2E+06	2.0E+07	1.1E+07	6.8E+06	2.9E+06	9.4E+05	7.6E+05	5135.3	5135.3	0
ncs	2598.0	1489.6	647.3	1.5E+06	1.1E+07	7.1E+06	3.7E+06	2.3E+06	1.0E+06	9.9E+05	4941.0	0.0	1
mcs	2815.9	1410.4	688.7	1.5E+06	1.4E+07	7.2E+06	4.4E+06	2.3E+06	5.3E+05	9.2E+05	4936.0	0.0	0
ct1	3696.6	1551.6	1194.6	2.6E+06	2.1E+07	1.1E+07	6.1E+06	3.4E+06	1.2E+06	1.0E+06	5377.3	5377.3	0
ct2	3032.6	1460.9	823.5	2.0E+06	1.6E+07	8.5E+06	4.9E+06	2.8E+06	8.2E+05	9.6E+05	5173.0	1825.7	0
ct3	3121.5	1497.9	816.4	1.8E+06	1.6E+07	7.9E+06	4.7E+06	2.6E+06	6.5E+05	9.8E+05	4986.3	547.0	0
ct4	2886.6	1419.7	727.2	1.6E+06	1.4E+07	7.4E+06	4.5E+06	2.4E+06	5.7E+05	9.3E+05	4921.0	166.7	0
ct5	2868.7	1432.7	705.9	1.6E+06	1.4E+07	7.4E+06	4.4E+06	2.4E+06	5.6E+05	9.3E+05	4940.3	60.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	169.6	158.1	11.9	1.2E+05	4.2E+05	7.6E+05	5.0E+05	2.6E+05	0.0E+00	1.1E+05	0.0	0.0	0
spb	100.1	70.1	17.1	1.4E+05	5.2E+05	5.8E+05	4.1E+05	1.6E+05	1.5E+04	8.3E+03	4709.3	4709.3	0
ncs	85.2	70.3	3.5	1.2E+05	3.1E+05	3.0E+05	6.4E+04	1.3E+05	1.1E+05	9.0E+03	4612.0	0.0	0
mcs	90.1	70.7	4.2	1.1E+05	4.0E+05	3.0E+05	1.7E+05	1.2E+05	7.3E+03	7.7E+03	4684.0	0.0	0
ct1	95.4	70.9	8.9	1.6E+05	4.0E+05	4.3E+05	2.6E+05	1.6E+05	7.6E+03	8.2E+03	4667.3	4667.3	0
ct2	90.1	70.4	5.6	1.3E+05	3.4E+05	3.4E+05	2.0E+05	1.4E+05	5.7E+03	7.9E+03	4677.0	1207.7	0
ct3	93.4	72.0	5.5	1.2E+05	4.3E+05	3.2E+05	1.8E+05	1.3E+05	1.0E+04	8.2E+03	4726.7	337.7	0
ct4	91.0	71.1	4.4	1.2E+05	4.1E+05	3.0E+05	1.7E+05	1.2E+05	8.3E+03	7.8E+03	4701.0	63.7	0
ct5	92.0	71.9	4.4	1.2E+05	4.1E+05	3.0E+05	1.7E+05	1.2E+05	8.4E+03	7.9E+03	4706.7	18.0	0

Table 4: The alignment results for Road traffic fine

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	1201.7	412.2	391.2	6.9E+06	4.6E+07	5.0E+07	4.1E+07	9.4E+06	0.0E+00	2.1E+05	0.0	0.0	0
spb	970.5	340.4	413.0	1.8E+06	9.5E+06	8.7E+06	6.7E+06	2.0E+06	4.8E+04	1.5E+05	527.3	527.3	0
cs	713.5	179.2	227.1	1.8E+06	9.5E+06	6.2E+06	1.5E+05	2.0E+06	4.1E+06	1.5E+05	530.3	0.0	5
mcs	746.9	177.5	226.0	1.8E+06	9.5E+06	7.4E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	534.0	0.0	0
ct1	744.8	175.0	227.8	1.8E+06	9.5E+06	7.5E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	536.7	536.7	0
ct2	745.3	176.2	226.5	1.8E+06	9.5E+06	7.4E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	534.7	0.0	0
ct3	745.6	175.6	226.7	1.8E+06	9.5E+06	7.4E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	533.0	0.0	0
ct4	744.6	175.7	226.3	1.8E+06	9.5E+06	7.4E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	534.0	0.0	0
ct5	749.2	178.3	227.0	1.8E+06	9.5E+06	7.4E+06	4.2E+06	2.0E+06	1.3E+06	1.5E+05	537.3	0.0	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	455.0	380.4	189.3	1.0E+06	3.4E+06	5.1E+06	3.7E+06	1.4E+06	0.0E+00	3.2E+05	0.0	0.0	0
spb	414.6	259.2	68.9	1.0E+06	3.5E+06	4.2E+06	3.0E+06	1.2E+06	2.9E+04	1.5E+05	9471.0	9471.0	0
cs	346.7	240.5	5.5	1.0E+06	3.3E+06	2.3E+06	1.6E+05	1.2E+06	1.0E+06	1.5E+05	8369.0	0.0	1
mcs	359.9	240.5	5.5	1.0E+06	3.3E+06	2.4E+06	1.2E+06	1.2E+06	1.3E+05	1.5E+05	8373.3	0.0	0
ct1	364.6	238.2	11.5	1.0E+06	3.4E+06	2.7E+06	1.3E+06	1.2E+06	1.4E+05	1.5E+05	8351.3	8351.3	0
ct2	358.0	237.5	6.7	1.0E+06	3.3E+06	2.5E+06	1.2E+06	1.2E+06	1.3E+05	1.5E+05	8355.0	916.7	0
ct3	356.3	237.1	5.6	1.0E+06	3.3E+06	2.4E+06	1.2E+06	1.2E+06	1.3E+05	1.5E+05	8366.7	19.7	0
ct4	356.1	236.8	5.6	1.0E+06	3.3E+06	2.4E+06	1.2E+06	1.2E+06	1.3E+05	1.5E+05	8360.3	0.0	0
ct5	359.2	239.5	5.6	1.0E+06	3.3E+06	2.4E+06	1.2E+06	1.2E+06	1.3E+05	1.5E+05	8346.3	0.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	780.4	591.3	195.9	2.3E+06	1.0E+07	1.3E+07	1.0E+07	3.4E+06	0.0E+00	5.2E+05	0.0	0.0	0
spb	500.0	191.1	175.8	1.4E+06	5.8E+06	5.9E+06	4.3E+06	1.5E+06	7.3E+04	1.5E+05	871.0	871.0	0
cs	440.1	176.8	92.5	1.4E+06	5.5E+06	4.1E+06	1.5E+05	1.5E+06	2.4E+06	1.5E+05	871.0	0.0	0
mcs	458.7	176.3	91.6	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	5.9E+05	1.5E+05	871.0	0.0	0
ct1	458.8	174.7	93.4	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	6.0E+05	1.5E+05	871.0	871.0	0
ct2	457.7	174.9	91.7	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	5.9E+05	1.5E+05	871.0	0.0	0
ct3	457.4	174.8	91.6	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	5.9E+05	1.5E+05	871.0	0.0	0
ct4	457.7	175.2	91.6	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	5.9E+05	1.5E+05	871.0	0.0	0
ct5	459.3	176.3	91.6	1.4E+06	5.5E+06	4.7E+06	2.6E+06	1.5E+06	5.9E+05	1.5E+05	871.0	0.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	615.3	522.4	180.0	1.3E+06	4.4E+06	6.8E+06	4.8E+06	1.9E+06	0.0E+00	4.7E+05	0.0	0.0	0
spb	326.5	178.9	66.5	9.8E+05	3.2E+06	4.0E+06	2.9E+06	1.1E+06	4.6E+04	1.5E+05	501.3	501.3	0
cs	294.8	168.7	21.9	9.8E+05	3.4E+06	2.5E+06	1.5E+05	1.1E+06	1.2E+06	1.5E+05	502.3	0.0	0
mcs	307.1	168.4	21.9	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.2E+05	1.5E+05	502.3	0.0	0
ct1	307.0	167.7	22.3	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.3E+05	1.5E+05	504.0	504.0	0
ct2	305.6	167.0	21.9	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.2E+05	1.5E+05	502.0	0.0	0
ct3	304.9	166.3	21.9	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.2E+05	1.5E+05	502.3	0.0	0
ct4	306.5	167.0	21.9	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.2E+05	1.5E+05	502.0	0.0	0
ct5	306.4	167.2	21.9	9.8E+05	3.4E+06	2.7E+06	1.4E+06	1.1E+06	2.2E+05	1.5E+05	502.7	0.0	0

Table 5: The alignment results for CCC2019

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	419.9	392.4	3.7	6.4E+04	7.9E+05	1.1E+06	8.3E+05	3.1E+05	0.0E+00	1.7E+05	0.0	0.0	0
spb	1835.2	126.7	1564.4	1.4E+05	1.8E+06	1.5E+06	1.2E+06	1.7E+05	4.3E+04	3.6E+04	76.7	76.7	0
cs	493.5	68.7	289.8	5.8E+04	7.0E+05	8.2E+05	7.0E+05	8.2E+04	4.0E+04	3.0E+04	83.3	0.0	0
mcs	496.6	66.6	299.8	5.7E+04	7.0E+05	6.0E+05	5.1E+05	7.9E+04	1.2E+04	2.9E+04	74.7	0.0	0
ct1	918.3	72.7	697.5	1.4E+05	1.8E+06	1.2E+06	1.0E+06	1.7E+05	3.8E+04	3.2E+04	74.7	74.7	0
ct2	489.1	73.4	307.0	1.1E+05	1.4E+06	9.5E+05	7.8E+05	1.3E+05	4.1E+04	3.2E+04	80.3	35.7	0
ct3	547.0	70.2	360.3	8.5E+04	1.0E+06	7.5E+05	6.2E+05	1.1E+05	2.8E+04	3.1E+04	77.0	19.0	0
ct4	593.4	69.5	386.4	7.7E+04	9.5E+05	6.8E+05	5.6E+05	9.9E+04	1.9E+04	3.0E+04	76.0	10.7	0
ct5	547.0	74.9	344.0	7.2E+04	8.8E+05	7.1E+05	5.9E+05	9.6E+04	2.0E+04	3.2E+04	77.3	5.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	55.5	53.3	0.5	1.2E+04	6.3E+04	1.3E+05	7.4E+04	5.2E+04	0.0E+00	2.6E+04	0.0	0.0	0
spb	175.3	113.5	48.5	3.9E+04	2.2E+05	2.2E+05	1.4E+05	6.9E+04	1.5E+04	3.1E+04	105.3	105.3	0
cs	40.3	32.0	5.5	7.4E+03	3.7E+04	8.3E+04	5.9E+04	1.9E+04	4.3E+03	1.3E+04	107.0	0.0	1
mcs	39.0	29.6	5.6	8.3E+03	4.4E+04	8.4E+04	6.1E+04	2.0E+04	3.1E+03	1.2E+04	105.3	0.0	0
ct1	241.4	100.1	80.7	1.2E+05	6.0E+05	4.1E+05	1.6E+05	1.6E+05	9.0E+04	4.3E+04	106.7	106.7	0
ct2	155.4	61.2	47.1	9.0E+04	4.4E+05	3.0E+05	1.1E+05	1.1E+05	7.6E+04	2.6E+04	103.0	48.0	0
ct3	126.3	58.8	34.0	6.6E+04	3.3E+05	2.4E+05	9.7E+04	8.9E+04	5.8E+04	2.4E+04	107.7	28.3	0
ct4	79.2	40.6	19.6	4.0E+04	2.0E+05	1.6E+05	7.7E+04	5.6E+04	3.2E+04	1.6E+04	103.3	19.7	0
ct5	57.6	40.2	10.5	1.8E+04	1.0E+05	1.2E+05	7.5E+04	3.3E+04	1.1E+04	1.6E+04	105.3	13.7	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	53.2	51.4	0.4	1.1E+04	4.9E+04	1.1E+05	6.2E+04	4.5E+04	0.0E+00	2.4E+04	0.0	0.0	0
spb	225.6	175.0	37.8	4.4E+04	2.1E+05	2.4E+05	1.4E+05	8.8E+04	1.2E+04	4.7E+04	132.0	132.0	0
cs	45.7	38.7	4.8	9.2E+03	3.2E+04	7.3E+04	4.9E+04	2.1E+04	2.4E+03	1.4E+04	128.7	0.0	0
mcs	65.5	51.5	8.2	1.7E+04	7.1E+04	1.0E+05	6.2E+04	3.4E+04	8.6E+03	2.0E+04	117.3	0.0	0
ct1	294.6	99.4	33.9	9.2E+04	4.1E+05	3.2E+05	1.2E+05	1.3E+05	7.2E+04	4.2E+04	123.7	123.7	0
ct2	122.6	86.7	19.8	5.1E+04	2.3E+05	2.1E+05	9.9E+04	8.2E+04	3.4E+04	3.5E+04	127.3	58.7	0
ct3	245.4	73.1	24.8	6.7E+04	2.9E+05	2.3E+05	8.3E+04	9.3E+04	5.6E+04	2.9E+04	127.3	34.7	0
ct4	243.3	73.7	23.3	6.4E+04	2.7E+05	2.3E+05	8.5E+04	9.0E+04	5.3E+04	2.9E+04	127.7	25.3	0
ct5	89.3	65.9	13.4	3.2E+04	1.4E+05	1.5E+05	7.6E+04	5.5E+04	2.1E+04	2.6E+04	120.7	15.7	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3.4	3.2	0.1	1.9E+03	5.0E+03	1.1E+04	6.4E+03	4.3E+03	0.0E+00	2.1E+03	0.0	0.0	0
spb	2.5	1.8	0.4	2.2E+03	6.2E+03	8.1E+03	5.9E+03	2.2E+03	2.2E+01	2.0E+01	44.7	44.7	0
cs	2.2	2.1	0.0	1.1E+03	2.2E+03	3.0E+03	9.3E+02	1.2E+03	9.2E+02	5.9E+01	54.7	0.0	0
mcs	1.3	1.1	0.0	1.1E+03	3.2E+03	2.6E+03	1.3E+03	1.2E+03	8.8E+01	3.5E+01	43.0	0.0	0
ct1	1.6	1.2	0.2	2.2E+03	6.4E+03	7.4E+03	5.1E+03	2.3E+03	1.7E+01	2.0E+01	45.7	45.7	0
ct2	1.4	1.1	0.1	1.7E+03	4.9E+03	4.7E+03	2.8E+03	1.8E+03	1.4E+02	2.0E+01	44.7	17.0	0
ct3	1.5	1.2	0.1	1.5E+03	4.3E+03	4.0E+03	2.3E+03	1.6E+03	1.5E+02	2.1E+01	48.7	10.7	0
ct4	1.3	1.1	0.1	1.4E+03	4.0E+03	3.5E+03	1.9E+03	1.5E+03	1.4E+02	2.1E+01	44.0	5.7	0
ct5	1.3	1.1	0.1	1.3E+03	3.6E+03	3.0E+03	1.7E+03	1.3E+03	8.2E+01	2.0E+01	43.0	4.7	0

Table 6: The alignment results for CCC2020-A

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	198.9	113.6	9.6	7.6E+05	4.1E+06	4146886	3.0E+06	1.2E+06	0.0E+00	57137.67	0.0	0.0	0
spb	169.3	41.6	88.1	2.8E+05	1.4E+06	974904.3	5.8E+05	2.9E+05	1.1E+05	16243.67	513.3	513.3	0
ncs	110.9	41.7	41.1	1.9E+05	8.2E+05	534837.3	1.0E+05	2.0E+05	2.3E+05	11390	699.0	0.0	1
mcs	114.1	39.9	40.8	1.9E+05	8.2E+05	603834.7	3.2E+05	2.0E+05	8.1E+04	10938.67	699.0	0.0	0
ct1	159.5	38.5	70.6	2.9E+05	1.3E+06	837985	4.3E+05	3.0E+05	1.1E+05	15054	559.7	559.7	0
ct2	142.5	40.5	57.9	2.5E+05	1.1E+06	739960.3	3.9E+05	2.6E+05	9.9E+04	12485	675.0	277.0	0
ct3	132.4	40.3	51.3	2.3E+05	1.0E+06	684939	3.6E+05	2.3E+05	9.2E+04	11895.67	684.0	146.3	0
ct4	127.7	40.5	48.5	2.2E+05	9.5E+05	662181.3	3.5E+05	2.2E+05	9.0E+04	11863	687.7	89.3	0
ct5	125.9	42.1	46.5	2.1E+05	9.2E+05	645612.7	3.4E+05	2.2E+05	8.5E+04	12546	699.3	30.0	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	243.3	232.7	4.6	1.1E+05	4.0E+05	735458	4.6E+05	2.7E+05	0.0E+00	130666.33	0.0	0.0	0
spb	51.9	29.6	14.6	8.7E+04	3.1E+05	358253.3	2.6E+05	9.0E+04	9.4E+03	1493	1336.0	1336.0	0
ncs	45.5	35.5	2.4	6.7E+04	2.2E+05	147135.7	1.1E+04	6.9E+04	6.7E+04	1821.67	1421.0	0.0	0
mcs	41.8	31.3	2.3	6.7E+04	2.2E+05	147436.7	7.4E+04	6.9E+04	3.9E+03	1751	1353.7	0.0	0
ct1	47.8	28.5	9.3	9.0E+04	3.0E+05	279705.7	1.8E+05	9.3E+04	6.6E+03	1493	1341.7	1341.7	0
ct2	43.7	30.0	4.7	7.7E+04	2.5E+05	193022.7	1.1E+05	7.9E+04	4.7E+03	1510.67	1348.7	424.0	0
ct3	42.9	30.6	3.6	7.2E+04	2.3E+05	169670	9.2E+04	7.4E+04	4.3E+03	1502	1356.3	190.0	0
ct4	41.6	30.2	2.9	7.0E+04	2.3E+05	157793.7	8.2E+04	7.2E+04	4.2E+03	1549	1354.0	84.0	0
ct5	41.5	30.8	2.5	6.8E+04	2.2E+05	149672.3	7.6E+04	7.0E+04	3.9E+03	1583	1359.7	23.7	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	297.9	285.1	5.0	1.3E+05	4.2E+05	830044.7	4.9E+05	3.4E+05	0.0E+00	170995	0.0	0.0	0
spb	56.7	34.8	14.6	8.7E+04	2.8E+05	352201.3	2.6E+05	9.0E+04	3.8E+03	1493	1625.0	1625.0	0
ncs	52.5	43.6	1.9	7.2E+04	1.8E+05	158217	1.3E+04	7.4E+04	7.1E+04	1581.33	1773.0	0.0	0
mcs	45.4	35.9	1.9	7.3E+04	1.8E+05	154232	7.7E+04	7.5E+04	2.3E+03	1526	1646.7	0.0	0
ct1	53.7	32.9	10.7	1.1E+05	2.9E+05	320070	2.1E+05	1.1E+05	3.2E+03	1493	1623.7	1623.7	0
ct2	49.7	35.5	5.3	8.7E+04	2.2E+05	218308	1.3E+05	8.9E+04	2.5E+03	1494.67	1650.3	574.3	0
ct3	46.5	35.0	3.3	7.9E+04	2.0E+05	180762.3	9.8E+04	8.1E+04	2.4E+03	1498.33	1642.7	229.3	0
ct4	46.4	35.6	2.8	7.7E+04	1.9E+05	171848.7	9.0E+04	7.9E+04	2.5E+03	1498.33	1649.3	135.3	0
ct5	45.1	35.3	2.1	7.4E+04	1.8E+05	159237	8.1E+04	7.6E+04	2.5E+03	1504.67	1646.0	42.3	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	210.8	202.2	3.9	1.1E+05	2.9E+05	626691.3	3.6E+05	2.6E+05	0.0E+00	128375.67	0.0	0.0	0
spb	75.3	56.8	11.2	1.0E+05	2.9E+05	371390	2.6E+05	1.1E+05	6.0E+03	1493	2662.3	2662.3	0
ncs	89.0	81.4	1.3	6.2E+04	1.4E+05	148565	2.7E+04	6.4E+04	5.7E+04	1751.33	3073.0	0.0	0
mcs	65.4	57.7	1.1	6.5E+04	1.5E+05	135615.7	6.5E+04	6.6E+04	4.1E+03	1500	2684.7	0.0	0
ct1	73.7	54.5	9.8	1.1E+05	2.7E+05	356360.3	2.4E+05	1.2E+05	3.9E+03	1493	2667.0	2667.0	0
ct2	70.4	56.6	5.4	9.1E+04	2.2E+05	245958.3	1.5E+05	9.3E+04	4.8E+03	1494	2671.7	1091.7	0
ct3	67.1	56.5	3.2	7.7E+04	1.8E+05	186134.7	1.0E+05	7.9E+04	3.3E+03	1494.33	2662.3	456.3	0
ct4	65.7	56.3	2.3	7.3E+04	1.7E+05	164746	8.6E+04	7.4E+04	4.0E+03	1497	2671.3	233.7	0
ct5	65.1	57.1	1.5	6.7E+04	1.6E+05	143485	7.1E+04	6.8E+04	4.1E+03	1493.33	2674.0	68.0	0

Table 7: The alignment results for CCC2020-B

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	206.3	139.1	9.5	6.7E+05	3.4E+06	3.8E+06	2.8E+06	9.6E+05	0.0E+00	7.1E+04	0.0	0.0	0
spb	697.9	129.8	408.3	1.3E+06	6.6E+06	4.4E+06	2.5E+06	1.3E+06	6.2E+05	1.3E+04	4227.0	4227.0	0
ncs	350.7	127.0	123.5	5.6E+05	2.8E+06	2.0E+06	8.9E+05	5.7E+05	5.0E+05	7.3E+03	4556.7	0.0	0
mcs	491.4	133.0	187.4	7.3E+05	3.6E+06	2.4E+06	1.2E+06	7.4E+05	4.5E+05	7.0E+03	4694.0	0.0	0
ct1	1101.9	146.9	602.8	1.8E+06	9.0E+06	5.5E+06	2.7E+06	1.8E+06	1.0E+06	1.5E+04	4585.0	4585.0	0
ct2	783.9	146.9	372.5	1.3E+06	6.5E+06	4.1E+06	2.0E+06	1.3E+06	8.7E+05	1.1E+04	4717.3	1966.3	0
ct3	700.9	143.4	319.7	1.1E+06	5.6E+06	3.6E+06	1.7E+06	1.1E+06	7.5E+05	1.0E+04	4711.7	1134.3	0
ct4	621.9	141.7	268.1	9.7E+05	4.9E+06	3.1E+06	1.5E+06	9.8E+05	6.4E+05	9.2E+03	4721.3	629.7	0
ct5	611.8	138.6	264.2	9.5E+05	4.7E+06	3.0E+06	1.5E+06	9.5E+05	6.2E+05	8.9E+03	4682.0	472.3	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	5437.1	5171.0	59.9	8.7E+05	6.7E+06	1.2E+07	7.3E+06	5.0E+06	0.0E+00	2.7E+06	0.0	0.0	0
spb	113.9	33.2	53.7	1.6E+05	1.1E+06	8.8E+05	6.7E+05	1.7E+05	4.6E+04	1.5E+03	1430.3	1430.3	0
ncs	121.4	71.5	24.2	1.1E+05	6.5E+05	3.3E+05	8.6E+04	1.3E+05	1.1E+05	2.3E+04	1441.3	0.0	0
mcs	69.3	32.6	11.7	9.8E+04	6.0E+05	2.3E+05	1.2E+05	9.9E+04	1.2E+04	1.7E+03	1426.0	0.0	0
ct1	108.8	32.7	37.7	1.6E+05	1.1E+06	6.6E+05	4.6E+05	1.7E+05	3.1E+04	1.5E+03	1425.0	1425.0	0
ct2	82.6	32.8	20.1	1.2E+05	7.6E+05	3.9E+05	2.4E+05	1.2E+05	2.5E+04	1.5E+03	1426.0	406.0	0
ct3	77.6	32.4	16.9	1.1E+05	7.1E+05	3.2E+05	1.9E+05	1.1E+05	1.8E+04	1.5E+03	1423.0	225.3	0
ct4	72.8	32.5	14.0	1.0E+05	6.4E+05	2.7E+05	1.5E+05	1.0E+05	1.5E+04	1.5E+03	1428.3	107.7	0
ct5	72.0	33.1	13.0	1.0E+05	6.2E+05	2.5E+05	1.4E+05	1.0E+05	1.4E+04	1.8E+03	1430.3	29.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3342.9	3196.6	31.5	5.3E+05	3.4E+06	7.1E+06	4.1E+06	3.0E+06	0.0E+00	1.8E+06	0.0	0.0	0
spb	117.7	47.8	47.3	1.6E+05	8.6E+05	7.8E+05	5.8E+05	1.7E+05	3.5E+04	1.5E+03	2194.3	2194.3	0
ncs	77.2	49.2	10.7	9.9E+04	4.5E+05	2.4E+05	4.2E+04	1.0E+05	9.4E+04	2.5E+03	2223.0	0.0	0
mcs	77.2	46.5	9.7	9.6E+04	4.8E+05	2.2E+05	1.2E+05	9.8E+04	9.3E+03	1.5E+03	2190.3	0.0	0
ct1	114.1	47.0	35.5	1.6E+05	8.6E+05	6.4E+05	4.5E+05	1.6E+05	2.4E+04	1.5E+03	2195.0	2195.0	0
ct2	92.1	46.9	19.9	1.2E+05	6.3E+05	3.9E+05	2.5E+05	1.3E+05	1.6E+04	1.5E+03	2190.0	849.3	0
ct3	83.8	46.7	14.1	1.1E+05	5.5E+05	2.9E+05	1.7E+05	1.1E+05	1.2E+04	1.5E+03	2186.3	341.7	0
ct4	81.1	47.2	11.8	1.0E+05	5.1E+05	2.6E+05	1.4E+05	1.0E+05	9.8E+03	1.5E+03	2196.3	158.3	0
ct5	78.5	46.7	10.3	9.8E+04	4.9E+05	2.3E+05	1.3E+05	1.0E+05	9.1E+03	1.5E+03	2186.3	51.7	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	872.4	835.2	9.4	3.2E+05	1.1E+06	2.1E+06	1.2E+06	9.4E+05	0.0E+00	5.0E+05	0.0	0.0	0
spb	115.3	79.0	24.6	1.3E+05	4.3E+05	5.4E+05	4.0E+05	1.4E+05	2.6E+03	1.5E+03	3430.3	3430.3	0
ncs	105.6	92.7	4.7	7.3E+04	2.0E+05	2.0E+05	5.8E+04	7.5E+04	6.8E+04	2.4E+03	3668.7	0.0	0
mcs	91.0	78.1	3.2	7.2E+04	2.2E+05	1.6E+05	8.0E+04	7.4E+04	3.0E+03	1.7E+03	3436.0	0.0	0
ct1	115.4	78.0	22.0	1.3E+05	4.3E+05	5.1E+05	3.7E+05	1.4E+05	2.2E+03	1.5E+03	3423.0	3423.0	0
ct2	103.0	79.2	11.2	1.0E+05	3.2E+05	3.1E+05	2.0E+05	1.1E+05	2.8E+03	1.5E+03	3436.0	1344.0	0
ct3	97.6	78.7	7.6	9.0E+04	2.8E+05	2.4E+05	1.5E+05	9.1E+04	2.4E+03	1.5E+03	3435.7	669.0	0
ct4	95.7	78.3	6.5	8.6E+04	2.6E+05	2.2E+05	1.3E+05	8.7E+04	2.8E+03	1.5E+03	3425.7	441.3	0
ct5	93.6	78.5	4.8	7.9E+04	2.4E+05	1.8E+05	1.0E+05	8.0E+04	3.0E+03	1.5E+03	3433.7	206.3	0

Table 8: The alignment results for CCC2020-C

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	906.6	826.8	14.6	6.1E+05	3.1E+06	4.1E+06	2.7E+06	1.4E+06	0.0E+00	3.9E+05	0.0	0.0	0
spb	1622.3	791.0	614.3	1.5E+06	7.5E+06	5.5E+06	3.2E+06	1.8E+06	5.0E+05	3.6E+05	2392.3	2392.3	0
ncs	838.5	523.3	189.9	5.7E+05	2.7E+06	2.8E+06	1.6E+06	7.5E+05	4.1E+05	2.0E+05	3379.3	0.0	0
mcs	853.6	506.5	192.6	5.8E+05	2.8E+06	2.9E+06	2.0E+06	7.7E+05	1.9E+05	2.1E+05	3340.3	0.0	0
ct1	1590.6	758.6	545.2	1.5E+06	7.2E+06	5.2E+06	2.9E+06	1.8E+06	5.3E+05	3.5E+05	2386.0	2386.0	0
ct2	1281.4	664.5	379.4	1.2E+06	5.7E+06	4.4E+06	2.5E+06	1.4E+06	5.0E+05	2.9E+05	2929.0	1320.7	0
ct3	1183.3	634.5	329.8	1.0E+06	4.9E+06	4.1E+06	2.4E+06	1.2E+06	4.2E+05	2.7E+05	3289.7	851.7	0
ct4	1045.1	591.3	259.7	7.7E+05	3.7E+06	3.4E+06	2.2E+06	9.9E+05	2.8E+05	2.5E+05	3315.3	321.0	0
ct5	1018.4	587.9	250.1	7.3E+05	3.5E+06	3.3E+06	2.1E+06	9.5E+05	2.5E+05	2.5E+05	3339.3	174.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	906.6	826.8	14.6	6.1E+05	3.1E+06	4.1E+06	2.7E+06	1.4E+06	0.0E+00	3.9E+05	0.0	0.0	0
spb	1622.3	791.0	614.3	1.5E+06	7.5E+06	5.5E+06	3.2E+06	1.8E+06	5.0E+05	3.6E+05	2392.3	2392.3	0
ncs	775.1	469.6	190.2	5.7E+05	2.7E+06	2.4E+06	1.3E+06	7.6E+05	4.1E+05	2.1E+05	2462.3	0.0	0
mcs	805.1	463.1	191.8	5.8E+05	2.8E+06	2.6E+06	1.6E+06	7.7E+05	1.9E+05	2.1E+05	2433.7	0.0	0
ct1	1627.2	762.1	563.1	1.5E+06	7.7E+06	5.3E+06	2.9E+06	1.8E+06	5.7E+05	3.5E+05	2382.3	2382.3	0
ct2	1202.6	613.8	350.0	1.1E+06	5.5E+06	4.0E+06	2.2E+06	1.3E+06	4.6E+05	2.8E+05	2433.7	1002.0	0
ct3	1042.1	563.0	277.7	7.9E+05	4.0E+06	3.2E+06	1.8E+06	1.0E+06	2.9E+05	2.5E+05	2404.3	324.7	0
ct4	1001.9	549.0	251.0	7.3E+05	3.7E+06	3.0E+06	1.8E+06	9.5E+05	2.6E+05	2.5E+05	2403.3	158.0	0
ct5	980.3	534.5	246.2	6.9E+05	3.5E+06	2.9E+06	1.7E+06	9.1E+05	2.4E+05	2.4E+05	2404.0	74.3	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	499.6	478.1	6.4	2.0E+05	6.8E+05	1.3E+06	7.5E+05	5.6E+05	0.0E+00	2.6E+05	0.0	0.0	0
spb	92.2	62.0	19.1	1.3E+05	4.5E+05	5.1E+05	3.6E+05	1.3E+05	1.5E+04	1.4E+03	2446.7	2446.7	0
ncs	100.7	88.5	3.5	7.7E+04	2.2E+05	1.9E+05	4.3E+04	7.9E+04	7.1E+04	2.2E+03	2860.3	0.0	0
mcs	77.9	65.1	3.3	7.9E+04	2.3E+05	1.7E+05	8.7E+04	8.0E+04	6.1E+03	1.5E+03	2504.7	0.0	0
ct1	91.8	61.3	15.7	1.3E+05	4.5E+05	4.5E+05	3.0E+05	1.3E+05	1.3E+04	1.4E+03	2444.7	2444.7	0
ct2	82.5	61.5	8.9	1.0E+05	3.4E+05	2.9E+05	1.8E+05	1.0E+05	9.4E+03	1.4E+03	2448.3	990.0	0
ct3	78.2	61.3	6.0	9.0E+04	3.0E+05	2.3E+05	1.3E+05	9.2E+04	7.7E+03	1.4E+03	2451.0	424.7	0
ct4	76.8	61.6	4.7	8.4E+04	2.7E+05	2.0E+05	1.1E+05	8.6E+04	7.9E+03	1.4E+03	2454.7	206.7	0
ct5	75.3	60.9	4.3	8.2E+04	2.6E+05	1.9E+05	9.7E+04	8.3E+04	7.2E+03	1.4E+03	2442.3	135.0	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	971.5	924.3	11.0	3.0E+05	1.3E+06	2.4E+06	1.4E+06	1.1E+06	0.0E+00	5.2E+05	0.0	0.0	0
spb	78.7	38.1	26.1	1.3E+05	5.8E+05	5.8E+05	4.3E+05	1.4E+05	9.5E+03	1.4E+03	2221.0	2221.0	0
ncs	57.0	40.8	5.4	8.1E+04	2.9E+05	1.8E+05	2.2E+04	8.2E+04	7.1E+04	1.4E+03	2319.7	0.0	0
mcs	57.1	39.1	5.3	8.1E+04	2.9E+05	1.8E+05	8.9E+04	8.2E+04	4.0E+03	1.4E+03	2270.0	0.0	0
ct1	78.0	37.4	20.5	1.3E+05	5.8E+05	4.7E+05	3.3E+05	1.3E+05	8.4E+03	1.4E+03	2220.3	2220.3	0
ct2	64.4	37.1	11.2	1.0E+05	4.3E+05	2.9E+05	1.8E+05	1.0E+05	8.3E+03	1.4E+03	2217.3	756.3	0
ct3	59.0	37.2	7.6	8.8E+04	3.6E+05	2.1E+05	1.2E+05	8.9E+04	5.3E+03	1.4E+03	2215.3	321.7	0
ct4	56.5	37.1	6.0	8.1E+04	3.3E+05	1.8E+05	9.1E+04	8.2E+04	5.4E+03	1.4E+03	2218.0	43.3	0
ct5	56.3	37.3	5.7	7.9E+04	3.2E+05	1.7E+05	8.6E+04	8.1E+04	5.1E+03	1.4E+03	2220.0	6.3	0

Table 9: The alignment results for CCC2020-D

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3840.1	3007.9	79.5	3.1E+06	2.7E+07	2.6E+07	1.7E+07	9.5E+06	0.0E+00	1.5E+06	0.0	0.0	0
spb	7259.1	1907.7	3750.0	4.1E+06	3.6E+07	1.4E+07	8.0E+06	4.5E+06	1.2E+06	9.2E+05	1621.3	1621.3	0
ncs	1704.8	618.5	633.2	1.4E+06	1.2E+07	5.0E+06	2.2E+06	1.6E+06	1.2E+06	2.9E+05	1592.0	0.0	6
mcs	1886.9	629.4	650.0	1.5E+06	1.2E+07	5.5E+06	3.2E+06	1.6E+06	6.4E+05	3.0E+05	1576.0	0.0	0
ct1	6999.7	1957.5	3106.6	4.5E+06	3.6E+07	1.4E+07	6.8E+06	5.0E+06	2.3E+06	9.5E+05	1597.7	1597.7	0
ct2	4582.3	1360.7	1865.1	3.1E+06	2.5E+07	1.0E+07	5.4E+06	3.5E+06	1.6E+06	6.4E+05	2151.0	826.3	0
ct3	3952.4	1220.6	1596.3	2.7E+06	2.2E+07	9.6E+06	5.1E+06	3.0E+06	1.5E+06	5.8E+05	2200.7	340.3	0
ct4	3471.3	1121.0	1304.0	2.3E+06	1.8E+07	8.3E+06	3.5E+03	2.8E+04	1.2E+06	5.2E+05	2176.5	135.5	0
ct5	2852.1	924.8	1068.5	2.0E+06	1.6E+07	7.5E+06	4.3E+06	2.2E+06	9.7E+05	4.3E+05	2191.0	85.7	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	2093.8	1992.0	20.6	3.7E+05	2.3E+06	4.5E+06	2.6E+06	1.9E+06	0.0E+00	1.0E+06	0.0	0.0	0
spb	79.6	36.2	29.4	1.1E+05	5.5E+05	5.3E+05	3.9E+05	1.1E+05	2.4E+04	1.3E+03	1367.3	1367.3	0
ncs	69.1	48.9	7.9	7.2E+04	3.1E+05	2.0E+05	4.4E+04	7.5E+04	7.6E+04	3.9E+03	1522.0	0.0	0
mcs	57.5	37.9	6.5	7.1E+04	3.1E+05	1.7E+05	9.3E+04	7.3E+04	6.2E+03	1.4E+03	1412.7	0.0	0
ct1	75.6	35.4	21.3	1.1E+05	5.2E+05	4.1E+05	2.8E+05	1.1E+05	1.4E+04	1.3E+03	1365.0	1365.0	0
ct2	64.1	36.4	12.3	8.8E+04	3.9E+05	2.6E+05	1.6E+05	9.0E+04	9.8E+03	1.3E+03	1378.3	503.0	0
ct3	61.5	37.3	9.7	8.2E+04	3.6E+05	2.2E+05	1.3E+05	8.3E+04	8.4E+03	1.3E+03	1401.3	243.0	0
ct4	61.2	38.3	8.6	7.7E+04	3.4E+05	2.0E+05	1.1E+05	7.9E+04	7.8E+03	1.3E+03	1413.3	132.0	0
ct5	58.4	37.0	7.7	7.5E+04	3.3E+05	1.9E+05	1.0E+05	7.7E+04	7.5E+03	1.3E+03	1410.0	67.0	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	1043.9	998.9	10.0	2.7E+05	1.2E+06	2.4E+06	1.3E+06	1.0E+06	0.0E+00	5.5E+05	0.0	0.0	0
spb	98.1	61.7	24.7	1.2E+05	4.6E+05	5.1E+05	3.7E+05	1.2E+05	1.7E+04	1.3E+03	2191.0	2191.0	0
ncs	117.2	101.1	6.0	7.3E+04	2.3E+05	2.2E+05	7.7E+04	7.7E+04	6.6E+04	4.7E+03	2678.0	0.0	2
mcs	83.0	67.8	4.6	7.4E+04	2.4E+05	1.7E+05	9.1E+04	7.6E+04	6.6E+03	1.5E+03	2293.7	0.0	0
ct1	96.0	60.8	19.5	1.2E+05	4.4E+05	4.3E+05	2.9E+05	1.3E+05	1.1E+04	1.3E+03	2200.0	2200.0	0
ct2	88.2	64.1	11.0	9.9E+04	3.3E+05	2.8E+05	1.7E+05	1.0E+05	9.0E+03	1.3E+03	2233.3	881.3	0
ct3	86.2	65.7	8.4	9.0E+04	2.9E+05	2.3E+05	1.3E+05	9.1E+04	7.9E+03	1.3E+03	2261.3	459.3	0
ct4	85.2	66.7	6.8	8.4E+04	2.7E+05	2.1E+05	1.2E+05	8.6E+04	7.6E+03	1.3E+03	2281.3	276.0	0
ct5	83.1	66.2	5.9	8.0E+04	2.6E+05	1.9E+05	1.0E+05	8.1E+04	7.1E+03	1.3E+03	2297.7	157.7	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	956.7	912.1	10.2	3.4E+05	1.2E+06	2.5E+06	1.4E+06	1.1E+06	0.0E+00	5.8E+05	0.0	0.0	0
spb	112.3	76.9	22.9	1.4E+05	5.0E+05	5.7E+05	4.2E+05	1.4E+05	7.4E+03	1.3E+03	2849.3	2849.3	0
ncs	115.4	101.6	4.4	7.8E+04	2.1E+05	2.0E+05	6.2E+04	8.0E+04	6.3E+04	2.4E+03	3201.3	0.0	0
mcs	93.1	79.1	4.0	7.8E+04	2.1E+05	1.7E+05	8.5E+04	8.0E+04	4.4E+03	1.7E+03	2880.0	0.0	0
ct1	109.2	75.1	18.2	1.5E+05	4.3E+05	4.9E+05	3.3E+05	1.5E+05	4.2E+03	1.3E+03	2846.3	2846.3	0
ct2	101.3	78.0	10.3	1.1E+05	3.1E+05	3.1E+05	1.9E+05	1.2E+05	4.9E+03	1.3E+03	2869.7	1187.3	0
ct3	99.3	79.9	7.5	1.0E+05	2.7E+05	2.5E+05	1.4E+05	1.0E+05	4.7E+03	1.3E+03	2887.3	608.7	0
ct4	96.0	78.6	6.0	9.1E+04	2.5E+05	2.1E+05	1.2E+05	9.3E+04	4.5E+03	1.3E+03	2873.3	336.7	0
ct5	92.3	76.2	5.4	8.8E+04	2.4E+05	2.0E+05	1.1E+05	9.0E+04	4.2E+03	1.5E+03	2863.3	235.7	0

Table 10: The alignment results for CCC2020-E

Model 0.05	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	1143.5	776.6	32.6	2.3E+06	1.5E+07	1.5E+07	9.9E+06	4.7E+06	0.0E+00	3.7E+05	0.0	0.0	0
spb	3356.1	692.2	1866.8	3.4E+06	2.3E+07	1.0E+07	5.7E+06	3.5E+06	9.3E+05	3.2E+05	2698.3	2698.3	0
ncs	1120.9	342.4	455.5	1.6E+06	1.1E+07	4.4E+06	1.4E+06	1.7E+06	1.4E+06	1.3E+05	3147.0	0.0	3
mcs	1280.9	355.7	466.7	1.6E+06	1.1E+07	5.0E+06	2.6E+06	1.7E+06	7.2E+05	1.3E+05	3137.0	0.0	0
ct1	3211.0	701.8	1592.7	3.5E+06	2.3E+07	1.0E+07	5.4E+06	3.6E+06	1.4E+06	3.1E+05	2694.0	2694.0	0
ct2	2526.7	618.0	1098.0	2.9E+06	1.9E+07	8.6E+06	4.3E+06	3.1E+06	1.2E+06	2.5E+05	3022.0	1309.3	0
ct3	1892.7	498.8	768.8	2.3E+06	1.5E+07	6.9E+06	3.4E+06	2.4E+06	1.1E+06	2.0E+05	3070.7	532.3	0
ct4	1762.6	478.4	701.0	2.1E+06	1.4E+07	6.3E+06	3.1E+06	2.2E+06	9.6E+05	1.9E+05	3109.3	283.7	0
ct5	1629.4	452.0	627.3	1.9E+06	1.3E+07	5.9E+06	3.0E+06	2.0E+06	9.0E+05	1.7E+05	3150.0	165.3	0

Model 0.20	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	3190.9	2963.4	34.4	5.5E+05	4.1E+06	7.3E+06	4.3E+06	3.0E+06	0.0E+00	1.6E+06	0.0	0.0	0
spb	97.2	30.7	46.1	1.3E+05	7.8E+05	6.6E+05	4.8E+05	1.3E+05	4.8E+04	1.4E+03	1521.0	1521.0	0
ncs	70.3	42.5	9.7	8.5E+04	4.4E+05	2.3E+05	5.4E+04	8.7E+04	9.4E+04	1.9E+03	1681.0	0.0	0
mcs	64.5	33.6	9.5	8.5E+04	4.4E+05	2.0E+05	1.0E+05	8.7E+04	1.1E+04	1.6E+03	1527.0	0.0	0
ct1	92.8	31.7	32.8	1.3E+05	7.3E+05	5.3E+05	3.7E+05	1.3E+05	3.1E+04	1.4E+03	1516.7	1516.7	0
ct2	75.6	33.0	17.5	1.1E+05	5.8E+05	3.3E+05	2.0E+05	1.1E+05	2.2E+04	1.4E+03	1525.0	466.0	0
ct3	70.0	33.2	13.7	9.6E+04	5.1E+05	2.7E+05	1.5E+05	9.7E+04	1.6E+04	1.4E+03	1523.0	207.7	0
ct4	66.1	33.1	11.3	9.0E+04	4.6E+05	2.3E+05	1.2E+05	9.1E+04	1.3E+04	1.4E+03	1527.7	109.0	0
ct5	64.9	33.3	10.1	8.6E+04	4.4E+05	2.1E+05	1.1E+05	8.8E+04	1.2E+04	1.6E+03	1522.3	22.7	0

Model 0.40	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	202.7	193.9	4.4	1.1E+05	3.2E+05	6.1E+05	3.6E+05	2.5E+05	0.0E+00	1.1E+05	0.0	0.0	0
spb	65.5	46.7	12.1	8.7E+04	2.7E+05	3.4E+05	2.4E+05	9.1E+04	5.6E+03	1.4E+03	1922.3	1922.3	0
ncs	71.3	64.3	1.2	6.5E+04	1.5E+05	1.5E+05	1.7E+04	6.7E+04	6.2E+04	1.6E+03	2164.3	0.0	0
mcs	59.7	51.5	1.1	6.5E+04	1.6E+05	1.4E+05	6.8E+04	6.7E+04	3.3E+03	1.5E+03	1941.0	0.0	0
ct1	66.7	48.3	9.1	1.0E+05	2.6E+05	3.1E+05	2.0E+05	1.0E+05	4.2E+03	1.4E+03	1922.3	1922.3	0
ct2	62.9	50.3	4.3	8.3E+04	2.0E+05	2.1E+05	1.2E+05	8.5E+04	3.6E+03	1.4E+03	1929.0	705.0	0
ct3	61.4	51.2	2.5	7.4E+04	1.8E+05	1.7E+05	9.3E+04	7.6E+04	3.4E+03	1.4E+03	1938.3	304.7	0
ct4	60.1	50.7	1.9	7.1E+04	1.7E+05	1.6E+05	8.2E+04	7.3E+04	3.3E+03	1.4E+03	1938.0	174.7	0
ct5	59.8	50.7	1.8	6.9E+04	1.7E+05	1.5E+05	7.9E+04	7.1E+04	3.3E+03	1.5E+03	1937.7	125.7	0

Model 0.80	Time			State space explored		Queue operations				LP number		Restart	Non-optimal
algorithm	total	heuristic	queue	states	arcs	total	insertion	removal	update	small LP	big LP		
reg	179.5	171.3	4.0	1.0E+05	2.9E+05	5.7E+05	3.3E+05	2.3E+05	0.0E+00	1.1E+05	0.0	0.0	0
spb	50.6	34.2	10.5	7.6E+04	2.4E+05	3.0E+05	2.2E+05	7.9E+04	2.1E+03	1.4E+03	1627.0	1627.0	0
ncs	52.4	46.7	0.8	5.5E+04	1.3E+05	1.2E+05	1.3E+04	5.6E+04	5.2E+04	1.5E+03	1838.7	0.0	0
mcs	44.4	37.7	0.8	5.5E+04	1.3E+05	1.1E+05	5.7E+04	5.7E+04	1.8E+03	1.4E+03	1643.7	0.0	0
ct1	50.5	35.3	7.5	8.7E+04	2.1E+05	2.6E+05	1.7E+05	9.0E+04	1.8E+03	1.4E+03	1627.7	1627.7	0
ct2	47.3	36.8	3.4	7.0E+04	1.7E+05	1.8E+05	1.0E+05	7.2E+04	1.9E+03	1.4E+03	1637.0	578.0	0
ct3	45.4	37.3	1.8	6.2E+04	1.4E+05	1.4E+05	7.4E+04	6.3E+04	1.6E+03	1.4E+03	1647.7	232.0	0
ct4	45.0	37.2	1.5	6.0E+04	1.4E+05	1.3E+05	7.0E+04	6.2E+04	1.7E+03	1.4E+03	1646.0	159.0	0
ct5	44.5	37.0	1.3	5.8E+04	1.3E+05	1.3E+05	6.5E+04	6.0E+04	1.8E+03	1.4E+03	1643.0	96.3	0

Acknowledgments

I would like to thank Prof.dr.ir. Wil van der Aalst, as I gained valuable insights through his fascinating courses at RWTH Aachen University and on Coursera, as well as the marvelous literature and paper in the field of process mining.

I also sincerely appreciate Professor Decker Stefan Josef, who kindly agreed to be the co-examiner of my thesis.

Moreover, I want to express my deep gratitude to my supervisor Dr.ir.Sebastiaan J. van Zelst. He gave me the chance to write this thesis and heavily mentored me throughout the whole writing process of my thesis. During his busy academic and research work, he always enlightens me with valuable ideas. I am grateful for his kind encouragement and patience whenever I made mistakes. Without his attentive and dedicated supervision, I would not have been able to finish this thesis smoothly.

Also, I would like to express my profound appreciation for my parents. Their unconditional love and support during my whole life, and I can always turn to them for help when I feel low. They not only supported me materially but also inspired me spiritually, as they embrace new ideas with an open mind and also encourage me to try.

Finally, I hope that submitting the thesis does not represent an end to my academic pursuit. I also hope I would always remain curious and never stop thinking.