Politecnico di Torino

Politecnico di Torino

Mathematical Engineering

A.a. 2025/2026

Graduation Session March 2026

Aligning Observed Timed Traces with Timed Stochastic Models

Supervisors:

Arianna Alfieri

Thomas Chatain

Paolo Ballarini

Candidate:

Sofia Bellotti

Abstract

Aligning observed and modeled behavior is a fundamental task in conformance checking. In this thesis, we propose a novel approach to alignments for stochastic timed process models, in which transitions occur after exponentially distributed random delays. Building on the traditional alignment framework of log moves and model moves, we introduce a likelihood-aware method that allows adjustments of observed timestamps, accounting for potential recording errors, to bring traces closer to the most likely behavior generated by the model according to its stochastic parameters. We define timed stochastic alignments between event log traces and Timed Stochastic Petri Nets by formulating an optimization problem that jointly balances CTMC path likelihoods and stamp-only temporal deviations. We first study the case of perfectly fitting models and show that the problem can be cast as a linear program, for which we derive structural properties of optimal solutions and efficient dynamic programming algorithms. We then extend the framework to models that do not perfectly fit the observed traces and to settings with partial-order alignments in extended free-choice nets. In this more general case, we prove that the alignment problem is NP-hard and propose exact and heuristic solution methods based on dynamic programming, Branch & Bound, and A* search.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Prof. Chatain, Prof. Ballarini, and Prof. Alfieri. In particular, I am deeply grateful to Prof. Chatain, with whom I worked closely every day at the Laboratoire des Méthodes Formelles. His guidance, support, and encouragement helped me integrate into the research environment and improve my French, even though there is still much to learn. I thank Prof. Ballarini for giving me the opportunity to participate in this research project and for his consistently constructive feedback, guidance, and support throughout. I am also thankful to Prof. Alfieri for agreeing to be my thesis advisor, for her kindness, and for providing support and valuable feedback, even from a distance. I am deeply grateful to my family for always supporting my choices. Although you would have preferred to have me closer, you did everything to ensure that I could make the most of this experience. To Bianca, I don't know you yet, but I already care for you infinitely. To Ari, my lifelong best friend, for being my constant and my certainty. A special thanks to Sara, who has been by my side since the very first day of university; without her, I would not have survived my first two months in Gif. Thanks to Giorgia, with whom I shared my entire undergraduate journey; I visited you across Europe, and I still can hardly believe that we now both live in Paris. I would also like to thank the friends I met at CentraleSupélec, who made my Erasmus experience unforgettable and strongly influenced my decision to return and complete my thesis. To my new friends connected through shared passions, particularly Lucia: we met through music, and you have become a constant presence in my daily life. Finally, I gratefully acknowledge the financial support of the Farman Institute at *ENS Paris-Saclay*, through project SToP, the *Laboratoire des Méthodes Formelles*, and the *Politecnico di Torino*, whose support made this research possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Process Mining utilizes data collected from process executions to discover *process models* and then analyze and evaluate them.
It is often described as a bridge between *data science*, which focuses on data-driven analysis, and *process science*, which traditionally relies on formal models without explicitly considering observed behaviour [1]. In the literature, process mining activities are commonly divided into three main categories:

- *Discovery*, which aims to construct process models in a data-driven manner.
- *Conformance*, which evaluates how well a given process model fits the observed data.
- *Enhancement*, which includes techniques that extend or improve a process model using additional information extracted from the data.

Historically, process models have been untimed and deterministic, as early modeling formalisms were primarily designed to capture the structural logic of processes. As a result, these models describe which sequences of activities can occur without accounting for how frequently they occur in practice. However, real-life processes rarely execute all possible paths equally: some variants occur frequently, while others represent rare exceptions caused by factors such as resource availability or operational constraints.
Deterministic models are unable to capture this variability and may therefore fail to accurately reflect how the process is actually performed.
To address this limitation, recent work has introduced stochasticity into process models so that they represent not only which sequences of events are possible, but also how frequently they occur in reality. Stochastic workflow nets achieve this by assigning a *weight* to each transition, which determines its probability of firing. These weights induce a (discrete) probability distribution over all runs of the

model, which can then be compared to observed trace frequencies using stochastic conformance measures [2, 3]. While significant progress has been made in modeling *untimed* stochastic processes [4, 5, 6, 7, 8], extending these ideas to the *timed* domain remains relatively unexplored.
In this thesis, we focus on the *temporal dimension* of events recorded in the form of *event logs* (that will be defined more precisley in Chapter 2). To capture both *how often* transitions occur and *when* they occur, we use Timed Stochastic Petri Nets. By interpreting the parameter of each transition as the rate of an exponential distribution, the model defines a continuous probability distribution over timed traces. This allows us to quantify not only how likely a sequence of actions is, but also how likely it is to observe these actions at specific points in time.
Timed stochastic models are therefore a natural choice for analyzing event logs that record when activities occur.
In this work, we introduce methods for aligning observed timed traces with these models. Trace-to-model alignment is a core task in conformance checking [9], essential not only for assessing model fitness, but also for applications such as model repair and improvement.

Structure of the Thesis

The first part of the thesis reviews some preliminaries.
In Chapter 2, we introduce basic concepts relevant to process mining, with a particular focus on *Petri nets*, and provide an introduction to conformance checking.
Chapter 3 presents the classical alignment problem and its time-extended variant.
The second part of the thesis presents our method.
Chapter 4 formalizes the stochastic timed alignment problem for perfectly fitting models. We cast it as a linear problem, analyze the structure of its solutions, and present efficient algorithms for computing them.
In Chapter 5, we extend the analysis to cases where an observed trace cannot be replayed exactly by the model.
Chapter 6 generalizes the framework to partial-order traces. We adapt the definition of alignment, derive a likelihood formula invariant under partial-order equivalence, and introduce the corresponding optimization problem. We also prove that this problem is NP-hard and propose both exact and approximate solution methods.
Chapter 7 describes the computational resolution of the optimization problems introduced in Chapters 4, 5, and 6.
Finally, Chapter 8 summarizes the main results of the thesis and discusses possible directions for future work.

Related work.

There is relatively little research focused on alignment analysis specifically for timed stochastic Petri nets. The relevant literature can be grouped into four main areas: stochastic conformance checking for untimed models, trace likelihood optimization in continuous-time Markov chains (CTMCs), temporal conformance checking without stochasticity, and model-free temporal stochastic conformance checking.

Stochastic Conformance Checking. Stochastic conformance checking extends traditional conformance measures by considering the probabilistic nature of both event logs and process models, treating them as distributions over traces. Leemans et al. [2] introduced *Earth Movers' Stochastic Conformance*, which uses the *Earth Movers' Distance* to compare event logs with stochastic process models by measuring the cost of transforming one trace distribution to another. Adriansyah et al. [10] proposed a cost-based fitness analysis for conformance checking, although time was not considered. Similarly, Li et al. [11] performed stochastic alignments without incorporating temporal information. More recently, Incerto et al. [12] proposed stochastic conformance checking leveraging variable-length Markov chains, but also without accounting for timing aspects. In recent work, Li, Polyvyanyy, and Leemans [13] studied the problem of matching an observed trace to a stochastic process model by formulating it as an optimization problem that jointly considers the likelihood of the model trace and its edit distance to the observed trace. Compared to their approach, which consider untimed stochastic models and discrete edit distances, our work adds a CTMC-based timing semantics, trades off likelihood and stamp-only temporal deviations, and treats partial-order alignments.

Trace Likelihood of CTMC. Computing the likelihood of trajectories in Continuous-Time Markov Chains (CTMCs) is fundamental for stochastic model validation. Perkins et al. [14] focused on maximizing the likelihood of CTMCs trajectories, given the initial state, end state, and a time bound. Related research by Grinberg and Perkins [15] and Levin et al. [16] developed methods to find the most likely sequence of states using the idea of *domination of sequences*, but without emphasizing sojourn times.

Temporal Conformance Checking. Temporal conformance checking addresses timestamped logs by measuring deviations in event timing. In [17], Chatain and Rino studied alignments for timestamp sequences, addressing the purely timed alignment problem with various metrics. Among these, they introduced the *stamp-only distance*, which we adopt as the time distance measure for this thesis.

Temporal Stochastic Conformance Checking. Recently, Richter et al. [18] proposed TADE (Temporal Activity Density Estimation), which estimates activity time densities via Gaussian kernels from logs and models them statistically for temporal stochastic conformance fitness. The approach avoids explicit model structures like Petri nets, relying instead on kernel density estimation, and thus cannot leverage CTMC-derived likelihoods from timed transitions or alignment-based explanations.

Contribution

This Master's thesis was carried out at the *Laboratoire des Méthodes Formelles (LMF)*, University *ENS Paris-Saclay*. The research was conducted under the supervision of Professor Chatain and Professor Ballarini, who contributed through scientific guidance, discussion, and critical revision of the work. The thesis includes work that is part of an unpublished manuscript.
The goal of this thesis is to address the gap identified in the literature by introducing a novel formulation of stochastic alignments for timed Petri nets.
Our approach explicitly incorporates timing information using CTMC semantics and jointly optimizes CTMC path likelihoods, timestamp deviations, and behavioral edit costs.
We introduce a formal framework for aligning timed traces from event logs with runs of timed stochastic workflow nets, extending existing approaches that were limited to untimed stochastic alignments. The framework is further generalized to handle partially ordered traces, allowing alignment of real-world processes where the exact order of some events may vary.
We formally prove that computing timed stochastic alignments is NP-hard and propose an efficient heuristic based on dynamic programming to compute approximate solutions in practice.
These contributions collectively provide a foundation for analyzing and understanding the temporal and stochastic characteristics of complex processes, bridging the gap between theoretical modeling and practical application.

Part I

Preliminaries

Chapter 2

Process Mining and Stochastic Petri Nets

Process mining is a family of techniques for analyzing event data recorded during the execution of business processes [19]. Its goal is to discover, monitor, and improve real processes by extracting knowledge from data collected in the form of *event logs*.
This chapter introduces the theoretical background used throughout the thesis. We first review the fundamental concepts of process mining, then present the Petri net formalism that forms the basis of the modeling framework adopted in this work, and finally introduce conformance checking.

2.1 Event Logs and Traces

An *event log* is a collection of recorded events, where each event relates to a specific *process instance*, called a *case*. Events have attributes such as a case identifier, an activity name, and a timestamp [20]. Each event represents one execution of an activity in a process; events belonging to the same case, ordered by time, form a *trace* that describes the execution of a single process instance.

Definition 2.1.1 (Alphabet, trace, timed trace, log.). We let $\Sigma = \{a, b, c, \dots\}$ denote the *alphabet* of an event log's activities and Σ^* the set of possibly infinite *traces* composed of activities in Σ, where $\varepsilon \in \Sigma^*$ represents the empty trace. A stochastic language over an alphabet Σ is a function $L : \Sigma^* \to [0,1]$ that provides the probability of the traces such that $\sum_{t \in \Sigma^*} L(t) = 1$. We denote $\hat{\sigma} = \langle (a_1, \theta_1), (a_2, \theta_2), (a_3, \theta_3), \dots, (a_n, \theta_n) \rangle \in (\Sigma \times \mathbb{R}_{>0})^*$ a *timed trace* where θ_i is the (absolute) time of occurrence of the i-th activity. We further denote $\delta_i = \theta_i - \theta_{i-1}$ (with the convention that $\theta_0 = 0$) the relative time of occurrence of action a_i w.r.t.

action a_{i-1}. Given a timed trace $\hat{\sigma}$ we denote $\bar{\hat{\sigma}} \in \Sigma^*$ its embedded (untimed) trace. An *event log* (or simply *log*) is a finite set of observed (timed) traces.

Definition 2.1.2 (Multi-sets). Let D be a finite domain. The set of multi-sets over D is defined as

$$\mathcal{B}(D) = \{X : D \to \mathbb{N}\}.$$

For a multi-set $X \in \mathcal{B}(D)$ and an element $d \in D$, the value $X(d)$ denotes the multiplicity of d in X.

2.2 Petri Nets

To formally represent process behaviour, we use Petri nets as the underlying modeling framework. Petri nets offer a precise graphical and mathematical language for describing concurrency, synchronization, and branching behaviour, and they can be naturally extended with timing and stochastic information [21]. For these reasons, they constitute the core formalism adopted throughout this thesis.

Definition 2.2.1 (Labelled Petri net [19]). A *labelled Petri net* is defined as a tuple

$$N = (P, T, F, \Sigma, \lambda, M_0),$$

where P and T are finite disjoint sets of places and transitions, respectively, and $F \subseteq (P \times T) \cup (T \times P)$ specifies the directed arcs of the net. The alphabet Σ contains the observable activity labels, while the labeling function $\lambda : T \to \Sigma \cup \{\tau\}$ associates each transition with either a visible label or the silent symbol τ. The initial state of the net is given by the marking $M_0 \in \mathbb{B}(P)$.

In this work, we consider only Petri nets with unit-weight arcs. This excludes the more general class of weighted Petri nets, in which arcs are assigned multiplicities through a function

$$F : (P \times T) \cup (T \times P) \to \mathbb{N}.$$

Definition 2.2.2 (Preset and postset). For a transition $t \in T$, the *preset* ${}^\bullet t$ is the set of places that have an arc directed toward t, while the *postset* $t^\bullet$ consists of the places reached by arcs originating from t:

$${}^\bullet t = \{p \in P \mid (p, t) \in F\}, \qquad t^\bullet = \{p \in P \mid (t, p) \in F\}.$$

The dynamic behaviour of a Petri net is described in terms of *markings*, which encode the current allocation of tokens over places. Intuitively, tokens represent active execution threads, and their distribution captures the global execution state of a process instance.

Definition 2.2.3 (Marking and state). A *marking* of a Petri net N is a function

$$M : P \to \mathbb{N},$$

where $M(p)$ denotes the number of tokens currently present in place p. The initial marking is denoted by M_0.

A transition $t \in T$ is said to be *enabled* at marking M if all its input places contain at least one token:

$$\forall p \in {}^\bullet t, \quad M(p) > 0.$$

We denote by

$$en(M) = \{t \in T \mid t \text{ is enabled at } M\}$$

the set of transitions enabled at marking M. When an enabled transition t fires, it produces a new marking M', written $M[t\rangle M'$, defined as

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet, \\ M(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t, \\ M(p) & \text{otherwise.} \end{cases}$$

A marking M' is *reachable* from a marking M if there exists a finite sequence of transitions $t_1, \ldots, t_n$ such that

$$M = M_1[t_1\rangle M_2[t_2\rangle \cdots [t_n\rangle M_{n+1} = M', \quad n \geq 1.$$

Two transitions t and t' are in *conflict* if they share an input place:

$${}^\bullet t \cap {}^\bullet t' \neq \emptyset$$

They are *concurrent* at a marking M if both are enabled at M and firing one does not disable the other.
A *run* of a Petri net N is a firing sequence

$$\bar{\sigma} = \langle t_1, \ldots, t_n \rangle$$

starting from the initial marking M_0. The corresponding *trace* is obtained by applying the labeling function:

$$\lambda(\bar{\sigma}) = \langle \lambda(t_1), \ldots, \lambda(t_n) \rangle \in \Sigma^*.$$

The set of all markings reachable from M_0 is denoted by $RS(N)$. The *reachability graph* of N is the labeled transition system

$$RG(N) = (RS(N), A),$$

where $A \subseteq RS(N) \times T \times RS(N)$ and $(M, t, M') \in A$ if and only if $M[t\rangle M'$. The *language* generated by a labelled Petri net N is

$$L(N) = \{\lambda(\bar{\sigma}) \in \Sigma^* \mid \bar{\sigma} \text{ is a run of } N\}.$$

Throughout this thesis, we focus on *safe* Petri nets, i.e., nets in which no reachable marking assigns more than one token to any place:

$$\forall M \in RS(N),\ \forall p \in P,\ M(p) \leq 1.$$

In addition, in this thesis we consider only *extended free choice Petri nets*, which restrict the structure of conflicts in a convenient way:

Definition 2.2.4 (Extended free choice Petri net [22, 23]). An *extended free choice* Petri net is a Petri net such that

$$\forall t, t' \in T, \quad {}^\bullet t \cap {}^\bullet t' \neq \emptyset \implies {}^\bullet t = {}^\bullet t'.$$

That is, any two transitions that share an input place must have exactly the same preset.

Definition 2.2.5 (Conflict cluster [24]). We define the equivalence relation

$$t \sim t' \iff {}^\bullet t \cap {}^\bullet t' \neq \emptyset.$$

A *conflict cluster* K is an equivalence class under $\sim$; that is, the maximal set of transitions with identical presets.

2.3 Timed Extensions of Petri Nets

When the timing of transitions is also considered, we obtain the notion of a *labelled timed Petri net*, following Wang [25]:

Definition 2.3.1 (Labelled timed Petri net [25]). A *labelled timed Petri net* (TPN) is a tuple

$$N_T = (P, T, F, \Sigma, \lambda, M_0, \delta),$$

where $(P, T, F, \Sigma, \lambda, M_0)$ is a labelled Petri net, and $\delta : T \to \mathbb{R}^+$ assigns a deterministic firing delay to each transition.

Intuitively, $\delta(t)$ represents the deterministic amount of time that must elapse after a transition t is enabled before it can fire.

2.4 Stochastic Petri Nets

Real-world processes often exhibit significant variability in their execution times. Manufacturing operations, medical treatments, and software systems are all subject to random delays and uncertain durations. Stochastic process models capture this variability by associating probabilistic timing behaviour with process executions. Stochastic Petri Nets extend classical Petri nets by integrating their structural modeling capabilities with the analytical framework of continuous-time Markov chains (CTMCs), thereby enabling quantitative performance evaluation [26]. This formalism allows one to capture both concurrency and synchronization, while supporting probabilistic timing analysis. The approach emerged independently in the late 1970s in the doctoral work of S. Natkin [27] and M. K. Molloy [28], leading to closely related formalisms that are now collectively known as *Stochastic Petri Nets*.

Definition 2.4.1 (Labelled Timed Stochastic Petri Net [26]). A *labelled timed stochastic Petri net* (TSPN) is a tuple

$$N_S = (P, T, F, \Sigma, \lambda, w, M_0),$$

where $(P, T, F, \Sigma, \lambda, M_0)$ is a labelled Petri net and $w : T \to \mathbb{R}^+$ assigns an exponential firing rate to each transition.

2.5 Conformance Checking: An Overview

In [19], conformance checking is described as the task of verifying whether the behavior recorded in an event log matches a given process model, and whether the model can represent the observed behavior. This assessment is commonly performed by considering four quality dimensions:

- *Fitness*: the model should be able to reproduce the behavior observed in the event log.
- *Precision*: the model should not allow for behavior that deviates significantly from what is observed in the event log.
- *Generalization*: the model should generalize beyond the observed behavior, allowing for plausible but unseen executions.
- *Simplicity*: the model should be as simple as possible, avoiding unnecessary complexity.

In this thesis, we primarily focus on the *fitness* dimension of conformance checking. In particular, we adopt *alignment-based* techniques, which provide a fine-grained and quantitative measure of deviations between observed behaviour and model behaviour. These techniques form the methodological foundation for the likelihood-aware alignment framework developed in Chapter 4.

Chapter 3

Alignments and Timed Conformance

Conformance checking techniques assess the degree to which the behaviour recorded in an event log conforms to the behaviour allowed by a process model. In the scope of this thesis, process models are represented as Petri nets. Given an event log and a process model, an algorithm is required to relate observed executions to model behaviour.
Two main classes of conformance checking algorithms have been proposed in the literature: *log replay* algorithms and *trace alignment* algorithms [9, 29]. Log replay techniques attempt to replay each trace of the event log on the model in order to determine whether it is executable. This approach can only indicate whether a trace fits the model or not, but provides little information about the nature of deviations.
To overcome this limitation, *alignments* were introduced. Alignment-based techniques explicitly relate events observed in the log to transitions executed in the model, thereby identifying deviations and quantifying them. As a result, alignments provide a fine-grained and diagnostic notion of fitness between a trace and a model. Following the seminal work of Van Der Aalst et al., Adriansyah defines in [30] an alignment between a recorded process execution and a process model as a stepwise comparison between executed activities in the trace and transitions fired in the model. Intuitively, if a trace perfectly fits the model, then each activity in the trace can be mimicked by firing a corresponding transition in the Petri net.

3.1 Moves and Legal Movement Sequences

Alignments are built by pairing events observed in an event log with transitions executed in the model. These pairings are referred to as *moves*. Let Σ be a finite

set of activities, let $\bar{\hat{\sigma}} \in \Sigma^*$ denote an untimed trace over Σ, and let N be a labelled Petri net over Σ. In the following, we present some classical definitions for moves and movement sequences, as introduced in [31].

Definition 3.1.1 (Move). A *move* is a pair (a, t), where a represents an activity from the log and t represents a transition of the model.

If an activity cannot be matched to a transition, or vice versa, the symbol $\gg$ is used to denote a *no move*.

Definition 3.1.2 (Legal and Illegal Moves). A move (a, t) is *legal* if it satisfies one of the following conditions:

- $a = \lambda(t) \in \Sigma$ (*synchronous move*);
- $a = \gg$ and $\lambda(t) \in \Sigma$ (*visible model move*);
- $a = \gg$ and $\lambda(t) = \tau$ (*invisible model move*);
- $a \in \Sigma$ and $t = \gg$ (*log move*).

All other pairs, such as $(\gg, \gg)$ or (a, t) with $\lambda(t) \neq a$, are considered *illegal*.

Legal moves define the elementary steps by which a trace and a model execution can be related. Sequences of such moves are used to explain how an observed trace may be aligned to the model.

Definition 3.1.3 (Legal Movement Sequence). A *movement sequence*

$$\gamma \in (\Sigma \cup \{\gg\} \times T \cup \{\gg\})^*$$

between a trace $\hat{\sigma}$ and a Petri net N is a sequence of moves such that:

- its projection onto the log, ignoring $\gg$, is a prefix of the observed trace:

$$\pi_1(\gamma) \downarrow_\Sigma \leq \hat{\sigma};$$

- its projection onto the model, ignoring $\gg$, is a prefix of a firing sequence of N:

$$\pi_2(\gamma) \downarrow_T \leq t, \quad t \in T^*.$$

A movement sequence is *legal* if all its moves are legal according to Definition 3.1.2.

Legal movement sequences characterize the space of admissible explanations relating an observed trace to a model execution. Among these sequences, alignments are obtained by selecting those that minimize a suitable distance or cost function.

3.2 Untimed Alignments

To quantify the degree of deviation between a trace and a model run, a notion of distance is required. A commonly adopted measure in this setting is the Levenshtein distance.

Definition 3.2.1 (Levenshtein Distance). The Levenshtein distance between two untimed traces is the minimum number of insertions, deletions, and substitutions required to transform one trace into the other.

In the context of alignments, insertions correspond to model moves, deletions correspond to log moves, and substitutions correspond to mismatches between activities and transitions.

Definition 3.2.2 (Alignment). Given a trace $\bar{\hat{\sigma}}$ and a Petri net N, an *alignment* is a complete run $\bar{\sigma}$ of N that minimizes the Levenshtein distance to $\bar{\hat{\sigma}}$ among all complete runs of N.

3.3 Timed Alignments

Classical alignments focus only on the ordering of activities and transitions, ignoring temporal information. While this is sufficient in purely structural analyses, real-world event logs usually include timestamps, which often carry valuable insights about process behaviour. Disregarding these timestamps can hide relevant deviations or anomalies. For example, even when the sequence of activities perfectly matches the model, unusually long or short delays between events may signal exceptional behaviour. This observation has motivated the development of alignment techniques that explicitly account for timing information. Chatain and Rino [17] proposed a first approach to incorporate time into the alignment problem by considering deterministic firing delays.

Definition 3.3.1 (Purely Timed Alignment Problem [17]). Given a timed trace

$$\hat{\sigma} = \langle (a_1, \hat{\theta}_1), \ldots, (a_n, \hat{\theta}_n) \rangle$$

and a timed Petri net N_T, a *timed alignment* is a run

$$\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n) \rangle$$

of N_T such that $\bar{\sigma} = \bar{\hat{\sigma}}$ (i.e., $\lambda(t_i) = a_i$ for all i), and the firing times $\theta_1, \ldots, \theta_n$ minimize a time-aware distance to $\hat{\sigma}$.

This formulation assumes a perfect structural correspondence between the trace and the model: it optimizes over timestamps only.
In this thesis, we consider the following distance function.

Definition 3.3.2 (Stamp-only Distance [17]). The stamp-only distance between a timed trace $\hat{\sigma}$ and a model run σ is defined as

$$d(\sigma, \hat{\sigma}) = \sum_{i=1}^{n} |\theta_i - \hat{\theta}_i|, \tag{3.1}$$

corresponding to the L_1 norm of the timestamp differences.

Purely timed alignments provide a principled way of explaining timestamp noise when the sequence of activities is fixed. However, they do not account for structural deviations, nor do they consider the stochastic nature of execution times.
Building on the notions of event logs, Petri nets, and alignments introduced so far, Chapter 4 extends classical alignment techniques by incorporating stochastic timing semantics and likelihood-based reasoning within the framework of Timed Stochastic Petri Nets.

Part II

Method

Chapter 4

Alignment of Timed Traces to Timed Stochastic Petri Nets

Building on the notions of event logs, Petri nets, and timed alignments introduced in Chapters 2 and 3, in this chapter we formalize the notion of alignment of observed timed traces to timed stochastic Petri nets as an optimization problem that accounts for both likelihood of traces and temporal deviations.

4.1 A Motivating Example

To motivate our approach, let us consider a company *invoice processing* process, which starts when an invoice is received and ends when the invoice is recorded. Its execution is structured into the following stages, each of which is represented, for simplicity, by a letter from the alphabet $\Sigma = \{a, b, c, c', d\}$:

- a: *capture invoice data*
- b: *verify purchase order* (PO)
- c: *check fraud (already known client)*
- c': *check fraud (new client)*
- d: *validate, approve, and pay the invoice*

Let us now assume that several executions of this process have been observed and

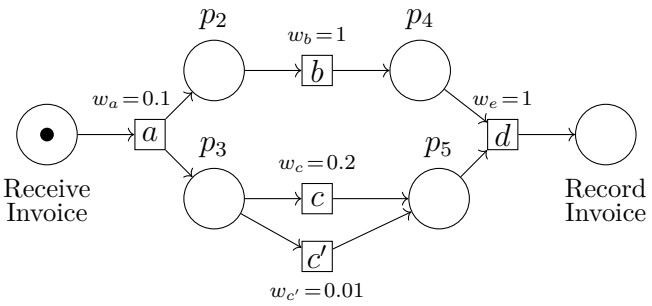

Figure 4.1: Timed stochastic Petri net model capturing the invoice processing log L.

recorded in the following event log L:

$$\langle (a,10.2), (b,11.3), (c,14.1), (d,14.9) \rangle$$
$$\langle (a,9.0), (b,10.2), (c,15.6), (d,16.2) \rangle$$
$$\langle (a,1.1), (b,10.2), (c,14.6), (d,15.5) \rangle$$
$$\langle (a,8.3), (b,9.1), (c,15.1), (d,16.2) \rangle$$
$$\langle (a,12.3), (c,12.5), (b,13.2), (d,19.1) \rangle$$
$$\langle (a,9.2), (b,10.2), (c',45.2), (d,45.9) \rangle$$

Each trace is a sequence of pairs (activity, timestamp), where timestamps denote absolute occurrence times. We can easily verify that the timed stochastic Petri net shown in Figure 4.1 is capable of reproducing all traces in L. After the firing of transition a, the net splits into two parallel branches: one enabling transition b and the other enabling either c or c'. As a result, transitions b and c (or c') can occur in any order, since they are concurrent, while exactly one of c and c' is executed, due to their conflict. Transition d is enabled only after both branches have completed, as it requires tokens from the places reached by b and by c or c'. Therefore, all traces in L, which start with a, end with d, and may have different orderings of the intermediate activities, correspond to valid firing sequences of the net.
The transitions of the model are assumed to follow exponential firing-time distributions, with rates $w_a = 0.1$, $w_b = 1$, $w_c = 0.2$, $w_{c'} = 0.01$, and $w_d = 1$, which characterize the random occurrence time of each activity. According to the model, any trace with the sequence of actions (a, b, c, d), (a, c, b, d), (a, b, c', d), or (a, c', b, d), provided that the timestamps are increasing, is possible. In this sense, all the sequences above are valid.
However, this form of validation is rather weak in terms of conformance checking, for two main reasons. First, it does not take into account the likelihood of the observed traces, which would provide a more informative measure of conformance.

Second, real event logs often contain deviations or recording errors, which in classical alignment-based approaches are handled using model moves and log moves [10]. A more robust conformance evaluation should therefore consider both the probability of the traces under the model and the possibility of deviations in the log.
Let us now reconsider our example. The rate associated with transition a is 0.1, which implies an expected firing delay of 10 time units, while transition b fires after 1 time unit on average. This generally agrees with the observed traces, except for the third one: $\langle(a, 1.1), (b,10.2), (c,14.6), (d,15.5)\rangle$. In this trace, action a occurs unusually early, at time 1.1. Although this is not in contradiction with the model (short dwell times are in fact the most likely under an exponential distribution), the long delay between a and b ($10.2 - 1.1 = 9.1$ time units) is highly unlikely.
While the model allows for this sequence, its likelihood is very low. An alternative explanation could be that the timestamp of a was recorded incorrectly and the activity actually occurred later, for example around time 9.5. After this correction, the resulting trace results far more likely under the model.
To move beyond mere feasibility and quantify *how well* an observed trace conforms to the model, we require a probabilistic semantics that assigns likelihoods to executions. This is provided by the stochastic semantics of timed stochastic Petri nets.

4.2 Stochastic Semantics of Timed Stochastic Petri Nets

We recall and adapt the stochastic semantics of TSPNs in a form suitable for alignment. The execution semantics of a TSPN can be mapped to a CTMC [32] by treating reachable markings as states and associating transition rates accordingly. An execution of a TSPN is described by a sequence

$$\rho = M_0 \xrightarrow{t_1,\theta_1} M_1 \xrightarrow{t_2,\theta_2} \cdots \xrightarrow{t_n,\theta_n} M_n,$$

where each θ_i denotes the absolute firing time of transition t_i. We write

$$\sigma = \langle(t_1, \theta_1), \ldots, (t_n, \theta_n)\rangle$$

for a run, and

$$\lambda(\sigma) = \langle(\lambda(t_1), \theta_1), \ldots, (\lambda(t_n), \theta_n)\rangle$$

for its labeled timed projection. The *timed language* of a TSPN N_S is

$$L(N_S) = \{\lambda(\sigma) \in \Sigma^* \times \mathbb{R}_{\geq 0} \mid \lambda(\bar{\sigma}) \in \mathrm{Path}(RG(N_S))\}.$$

When a marking is entered, each enabled transition independently samples a firing delay from its exponential distribution. The transition with the smallest sampled delay fires first, determining both the next marking and the sojourn time in the current state. Thanks to the memoryless property of exponential distributions, the minimum of these delays is itself exponentially distributed, with a rate equal to the sum of the rates of all enabled transitions. This property underlies the Markovian semantics of TSPNs.

4.2.1 Likelihood of Timed Runs

Since a TSPN induces a CTMC whose states correspond to reachable markings [26], the likelihood of a run $\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n) \rangle$ is

$$\ell(\sigma) = \prod_{i=1}^{n} w_{t_i} \, e^{-W_{t_i}(\theta_i - \theta_{i-1})}, \tag{4.1}$$

where W_{t_i} denotes the sum of firing rates of all transitions enabled in the marking M_i reached after the partial run $\langle (t_1, \theta_1), \ldots, (t_i, \theta_i) \rangle$, i.e.,

$$W_{t_i} = \sum_{j \in en(M_i)} w_{t_j}$$

Intuitively, each factor in (4.1) accounts for the probability density of firing transition t_i after a waiting time $\delta_i = \theta_i - \theta_{i-1}$, given the competition with all other enabled transitions.

4.3 Likelihood-aware Alignment of Timed Traces to TSPNs

We will now define the alignment problem between an observed timed trace and a timed stochastic process model as an optimization problem with an objective function that combines the likelihood of a model run with its distance to the observation.

In this thesis, we focus on timestamps and timed stochastic models. Accordingly, in this chapter we do not address the problem of aligning observed traces to an under-fitting model—i.e., a model whose language does not contain the observation, which is typically handled by inserting log moves and model moves as in [10]. Not-perfectly fitting models will be considered in Chapter 5.

Therefore, in our definition of alignment of an observed trace $\hat{\sigma} = \langle (a_1, \hat{\theta}_1) \ldots (a_n, \hat{\theta}_n) \rangle$ we assume that the model language contains a run $\sigma = \langle (t_1, \hat{\theta}_1) \ldots (t_n, \hat{\theta}_n) \rangle$ such that $\lambda(t_i) = a_i$ for all i, i.e. the model can replay exactly the observed timed trace.

More specifically, our point is that even when the model can perfectly replay the observed trace $\hat{\sigma} = \langle (a_1, \hat{\theta}_1) \dots (a_n, \hat{\theta}_n) \rangle$, this exact replay may not be the optimal alignment once the likelihood of traces is taken into account.
Motivated by this, our goal is to extend the deterministic timed alignment problem 3.3.1 to incorporate stochasticity. The extended problem must combine two aspects: the likelihood of the model trace and its distance from the observed trace. The relative importance of these two aspects is controlled by the parameters α and β. As a metric for the distance between observed timestamps and those of the aligned model run, we use the *stamp-only distance (3.1)*, which simply corresponds to the L_1 norm between the vectors of timestamps $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ and $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \dots, \hat{\theta}_n)$:

$$\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_{L_1} = \sum_{i=1}^{n} |\theta_i - \hat{\theta}_i|$$

Definition 4.3.1 (Alignment of a Timed Trace to a TSPN). Given an observed trace $\hat{\sigma} = \langle (a_1, \hat{\theta}_1) \dots (a_n, \hat{\theta}_n) \rangle$ and a TSPN $N_S = (P, T, F, \Sigma, \lambda, w, M_0)$, an *alignment* of $\hat{\sigma}$ to N_S is a run $\sigma = \langle (t_1, \theta_1) \dots (t_n, \theta_n) \rangle$ of N_S such that $\bar{\sigma} = \bar{\hat{\sigma}}$ (i.e. $\lambda(t_i) = a_i$ for all i) and the firing times $\theta_1, \dots, \theta_n$ minimize the objective function

$$\alpha \left(-\ell(\sigma)\right) \; + \; \beta \, \|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_{L_1} \tag{4.2}$$

where $\ell(\sigma)$ denotes the likelihood of σ, as defined in (4.1), and $\alpha, \beta \geq 0$ are tunable parameters such that $\alpha + \beta = 1$. The choice of α and β balances the emphasis between the likelihood and the distance to observed times.

In likelihood-minimization problems, it is common to work with the log-likelihood rather than the likelihood itself, since the logarithm is strictly increasing, thus the likelihood and the log-likelihood attain their maximum at the same point, and it turns products of probabilities into sums, which are easier to handle analytically and numerically [33]. The corresponding log-likelihood for a model trace σ is given by

$$\begin{aligned} \mathcal{L}(\sigma) &= \log \ell(\sigma) \\ &= \log \left(\prod_{i=1}^{n} w_{t_i} \, e^{-W_{t_i}(\theta_i - \theta_{i-1})} \right) \\ &= \sum_{i=1}^{n} \log w_{t_i} - W_{t_i}(\theta_i - \theta_{i-1}). \end{aligned} \tag{4.3}$$

Notice that the constant term $\sum_{i=1}^{n} \log w_{t_i}$ does not depend on the timestamps and therefore does not affect the optimization. Consequently, the alignment depends only on exit rates W_{t_i}, not on individual transition rates. Thus, our optimization problem can be formulated as the following.

Definition 4.3.2 (Timed Stochastic Alignment Problem).

$$\begin{aligned} \min_{\theta_1,\dots,\theta_n} \quad & \alpha \sum_{i=1}^{n} W_{t_i}(\theta_i - \theta_{i-1}) + \beta \sum_{i=1}^{n} \left|\theta_i - \hat{\theta}_i\right| \\ \text{subject to} \quad & \theta_i \leq \theta_{i+1}, \quad \forall i = 1, \dots, n-1, \\ & \theta_n \geq \hat{\theta}_n. \end{aligned} \tag{4.4}$$

where the constraint $\theta_n \geq \hat{\theta}_n$ ensures that the optimized trace duration is at least as long as the observed trace, preventing premature termination of the trace when α dominates the objective. Without this constraint, the optimization would collapse all timestamps to their minimum feasible value, artificially shortening the trace duration to maximize likelihood.
The solution depends on the relative values of α and β and can, for example, be computed using the *simplex method*, which explores the vertices of the feasible polyhedron to identify the optimal point.

Proposition 1. *For every α and β, there exists a solution of the Timed Stochastic Alignment Problem 4.3.2 where all the firing times $\theta_i^\star$ are taken in the set of observed timestamps $D = \{\hat{\theta}_1, \dots, \hat{\theta}_n\}$.*

Proof. The problem 4.3.2 can be easily reformulated as a linear programming (LP) problem, a class of optimization problems in which a linear objective function is minimized subject to the variables lying in a polyhedron [34]. To do this, we introduce non-negative slack variables u_i to replace the absolute value terms, obtaining the equivalent formulation

$$\begin{aligned} \min_{\theta_1,\dots,\theta_n,\, u_1,\dots,u_n} \quad & \alpha \sum_{i=1}^{n} W_{t_i}(\theta_i - \theta_{i-1}) + \beta \sum_{i=1}^{n} u_i \\ \text{subject to} \quad & \theta_i \leq \theta_{i+1}, \quad i = 1, \dots, n-1, \\ & \theta_n \geq \hat{\theta}_n, \\ & u_i \geq 0, \quad i = 1, \dots, n, \\ & u_i \geq \theta_i - \hat{\theta}_i, \quad i = 1, \dots, n, \\ & u_i \geq -(\theta_i - \hat{\theta}_i), \quad i = 1, \dots, n. \end{aligned} \tag{4.5}$$

Thus, (4.5) is a linear program in the variables $(\theta_1, \dots, \theta_n, u_1, \dots, u_n)$, since all constraints are linear and define a polyhedron, i.e., the intersection of a finite number of closed half-spaces and hyperplanes in $\mathbb{R}^{2n}$.
As stated in classical linear programming theory (see, e.g., [35]), for a linear programming minimization problem where the feasible set is a nonempty polyhedron, either the optimal cost is $-\infty$ or there exists an optimal solution which is an extreme

point. In problem 4.3.2 the observed trace can be replayed and the corresponding model-trace is a feasible solution with finite cost and the feasible set is a nonempty polyhedron (since the constraints are linear inequalities); thus we can conclude that there exists an optimal solution which is an extreme point of the feasible set. □

Implementation To solve the optimization problem, we implemented a function that takes as input the observed action timestamps, the transition rates W, the matrices defining the linear constraints, and a grid of α values. For each value of α, the function returns an array containing α and β, the optimal timestamps $\boldsymbol{\theta}$, the timestamp-based distances, the value of the objective function at the optimum, and the optimization method used. Specifically, the function relies on MATLAB's *linprog* to solve the underlying linear program. A complete listing of the function is provided in Appendix (A.1).

Example

Let us consider the invoice workflow depicted in Figure 4.1 and suppose we observe the trace

$$\hat{\sigma} = \langle (a, 1.1), (b, 10.2), (c, 14.6), (d, 15.5) \rangle.$$

Aligning this trace to the model N_S consists of finding a model run

$$\sigma = \langle (a, \theta_1), (b, \theta_2), (c, \theta_3), (d, \theta_4) \rangle$$

with firing times $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$ that optimally balance two objectives: the timestamp-based distance to the observed timestamps $\hat{\boldsymbol{\theta}} = (1.1, 10.2, 14.6, 15.5)$ and the likelihood of the trace under the model. These criteria are weighted by the tunable parameters α (for the likelihood) and β (for the timestamp deviation).

Likelihood term The likelihood of a run can be directly derived from the CTMC induced by the TSPN depicted in Figure 4.1. This CTMC has states corresponding to the following markings:

$$M_1 = \{p_1\}, \quad M_2 = \{p_2, p_3\}, \quad M_3 = \{p_3, p_4\},$$
$$M_4 = \{p_2, p_5\}, \quad M_5 = \{p_4, p_5\}, \quad M_6 = \{p_6\}.$$

The resulting CTMC is illustrated in Figure 4.2. The exit rate of each state coincides with the total firing rate of the transitions enabled in that marking, so the likelihood of a particular run can be expressed in terms of the exponential waiting-time distributions associated with these exit rates. Each marking M_i can also be represented as a vector over the places $(p_1, p_2, p_3, p_4, p_5, p_6)$:

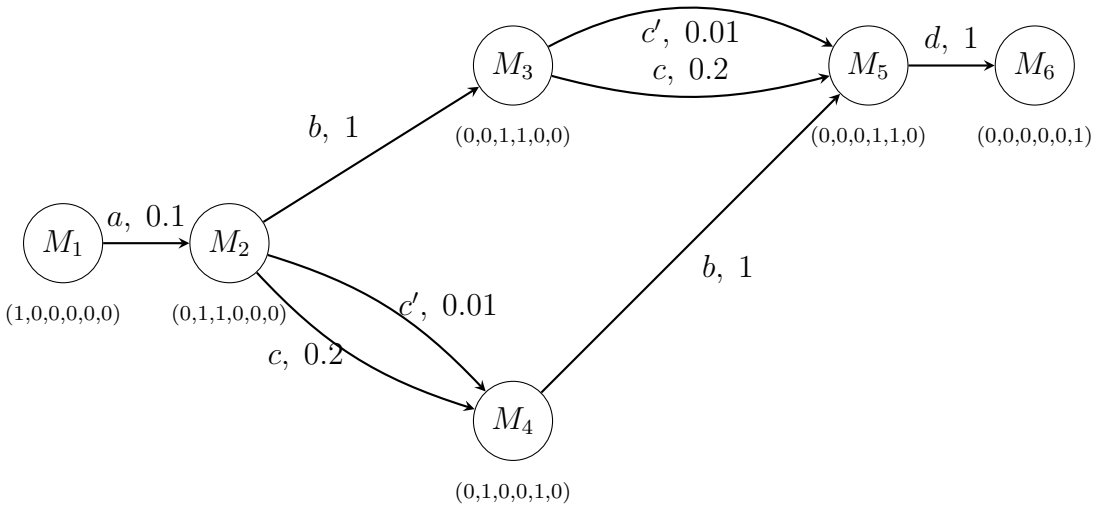

Figure 4.2: CTMC induced by the TSPN of figure 4.1

State		Rate
M_1	(1,0,0,0,0,0)	0.1
M_2	(0,1,1,0,0,0)	1.201
M_3	(0,0,1,1,0,0)	0.201
M_4	(0,1,0,0,1,0)	1
M_5	(0,0,0,1,1,0)	1
M_6	(0,0,0,0,0,1)	0

Each row corresponds to a marking of the TSPN. The exit rate of a state is the sum of the firing rates of all transitions enabled in that marking. For example, in $M_2 = (0,1,1,0,0,0)$, transitions b, c, and c' are enabled, so the exit rate of M_2 is $w_b + w_c + w_{c'}$.

Resolution We solved this optimization problem using a grid of 10,000 α values to obtain fine-grained estimates of the points at which the optimal solution changes. The solver converged to the optimum in 4.6578 seconds, using the dual-simplex method. Table 4.1 reports the resulting timestamps for selected α values: the commonly used values 0, 0.25, 0.5, 0.75, and 1, as well as the critical α values where the optimal solution changes (0.4760, 0.5560, 0.9520), as will be discussed in more detail in the following paragraph.

Interpretation of the results When $\beta = 1$, the optimizer minimizes only the timestamp deviation, so the resulting transition firing times exactly match the

α	β	$\boldsymbol{\theta}$
0.000,0	1.000,0	[1.1, 10.2, 14.6, 15.5]
0.250,0	0.750,0	[1.1, 10.2, 14.6, 15.5]
0.476,0	0.524,0	[10.2, 10.2, 14.6, 15.5]
0.500,0	0.500,0	[10.2, 10.2, 14.6, 15.5]
0.556,0	0.444,0	[10.2, 10.2, 15.5, 15.5]
0.750,0	0.250,0	[10.2, 10.2, 15.5, 15.5]
0.952,0	0.048,0	[15.5, 15.5, 15.5, 15.5]
1.000,0	0.000,0	[15.5, 15.5, 15.5, 15.5]

Table 4.1: Optimal timestamps $\boldsymbol{\theta}$ for different weightings α and β.

observed timestamps for the corresponding actions. When $\alpha = 0.5$ and $\beta = 0.5$, the optimizer gives equal weight to the likelihood term and the timestamp deviation penalty. The resulting timestamps align with our intuition: since the rate of transition a is low, it is expected to occur later than the observed value 1.1. Indeed, at $\alpha = 0.5$, the model shifts transition a forward in time so that it fires together with transition b (at time 10.2), while the remaining transitions retain their observed timestamps due to the equal weighting. As α becomes dominant, for example at $\alpha = 0.75$, the likelihood term drives the optimization. The model delays transition c as well, firing it later to increase its contribution to the likelihood. In the extreme case $\alpha = 1$, the timestamp-distance term is ignored entirely. The optimizer maximizes the likelihood by letting all transitions fire at the latest observed timestamp, resulting in all timestamps coinciding with the final observed value.

We remark that this structure relies crucially on the use of the log-likelihood in the objective, which yields a linear contribution in the firing times for exponential holding times. As a result, the overall cost function in the Timed Stochastic Alignment Problem 4.3.2 is linear, and the problem is a linear program. If the likelihood itself were used instead, the optimization would become nonlinear, and the constraint that optimal solutions lie at extreme points of the feasible polyhedron would no longer apply; in that case, the optimal timestamps would typically vary smoothly with different α and β.

A key qualitative feature of Table 4.1 is that, between consecutive breakpoint values of α, the optimal timestamps remain constant. Across the entire interval $\alpha \in [0,1]$, only four distinct optimal solutions occur. This piecewise-constant behaviour is not accidental, but a direct consequence of the linear programming structure of the Timed Stochastic Alignment Problem 4.3.2.

By Proposition 1, every optimal solution can be chosen with timestamps drawn from the finite set $D = \{\hat{\theta}_1, \ldots, \hat{\theta}_n\}$. Since the objective function depends linearly

on (α, β), changes in the optimal solution can occur only at finitely many parameter values where two extreme points exchange optimality.
This structural property admits a natural geometric interpretation, which we now make explicit.

4.3.1 Geometric interpretation of the parametric solution

The piecewise-constant behaviour observed in Table 4.1 can be understood geometrically. The Timed Stochastic Alignment Problem 4.3.2 is a linear program in the variable $\boldsymbol{\theta}$, and its objective function depends linearly on the parameters α and β (with $\alpha + \beta = 1$). Since the feasible region is defined by linear constraints, it is a polyhedron, and therefore optimal solutions are attained at its extreme points.
The objective can be written as

$$f_\alpha(\boldsymbol{\theta}) = \alpha f_1(\boldsymbol{\theta}) + (1 - \alpha) f_2(\boldsymbol{\theta}),$$

where $f_1(\boldsymbol{\theta})$ represents the likelihood contribution and $f_2(\boldsymbol{\theta})$ the timestamp-deviation term. Both are linear functions of $\boldsymbol{\theta}$. For any fixed extreme point $\boldsymbol{\theta}^{(e)}$, the value $f_\alpha(\boldsymbol{\theta}^{(e)})$ is therefore an affine function of α. Because the feasible polyhedron has only finitely many extreme points, the optimal value is obtained by taking the minimum among finitely many affine functions of α. As a consequence, the optimal solution remains constant over intervals of α and changes only at finitely many breakpoint values. These breakpoints correspond to parameter values at which two extreme points yield the same objective value and the optimizer switches from one to another. In particular, optimal timestamps can always be chosen among the observed timestamps, and the alignment changes only at specific critical values of α. This explains why only a small number of distinct optimal solutions appear across the whole interval $\alpha \in [0,1]$.

4.3.2 Likelihood-only maximization

When $\alpha = 1$, the distance term in the objective function of the Timed Stochastic Alignment Problem 4.3.2 is ignored, and the optimization problem reduces to maximizing the likelihood of the model run. The resulting behaviour is closely related to the boundary problem studied by Perkins [14].
In that setting, for a given trajectory

$$\boldsymbol{U} = (s_0, \delta_0, s_1, \delta_1, \ldots, \delta_{n-1}, s_n)$$

of a CTMC with exit rates W, the maximum-likelihood waiting times $\delta_1^*, \ldots, \delta_n^*$, subject to the constraint $\sum_{i=1}^{n} \delta_i \leq B$ for a fixed time horizon B, allocate all available time to the state with the smallest exit rate.

Our setting differs in that the final observed timestamp corresponds to the time at which the process reaches the final state. Consequently, the maximum-likelihood solution allocates all available time to the intermediate state with the smallest exit rate, while the final state, whose exit rate is zero, does not contribute to the allocation. The constraint therefore becomes $\sum_{i=1}^{n-1} \delta_i \leq B$, which is equivalent to solving the Perkins problem for the truncated trajectory

$$\boldsymbol{U} = (s_0, \delta_0, s_1, \delta_1, \ldots, \delta_{n-2}, s_{n-1}).$$

This can be shown by contradiction. Let

$$j = \arg \min_{1 \leq i \leq n-1} w_{s_i},$$

and suppose that there exists an index $i \neq j$ such that $\delta_i > 0$. Transfer a small amount of time $\epsilon > 0$ from δ_i to δ_j, while preserving the constraint $\sum_{i=1}^{n-1} \delta_i = \theta_{n-1} \geq \hat{\theta}_{n-1}$. The resulting change in the objective function is

$$\Delta = \epsilon\Big(w_{s_j} - w_{s_i}\Big) < 0,$$

since w_{s_j} is minimal by definition. This strictly decreases the objective value, contradicting the optimality of the original allocation. Therefore, in any optimal solution, all available time must be assigned to the state with the smallest exit rate, i.e.

$$\delta_j = \hat{\theta}_n, \qquad \delta_i = 0 \quad \forall i \neq j.$$

Let us now reconsider the observed trace $\hat{\sigma}$ from Example 4.3. We have already seen that, when $\alpha = 1$, the optimization problem yields the solution

$$\sigma = \{(a,15.5), (b,15.5), (c,15.5), (d,15.5)\},$$

in which all timestamps collapse to the final observed time.
This behaviour can also be directly justified using the result derived above. In this example, the slowest state of the CTMC shown in Figure 4.2 is M_1, with minimal rate $w_{\min} = w_a = 0.1$. Thus all available time is assigned to M_1:

$$\delta^*_{M_1} = 15.5, \qquad \delta^*_{M_2} = 0, \qquad \delta^*_{M_3} = 0, \qquad \delta^*_{M_5} = 0,$$

which yields exactly the run $\sigma = \{(a,15.5), (b,15.5), (c,15.5), (d,15.5)\}$, in accordance with the optimal solution shown in Table 4.1.

4.4 Dynamic Programming Solution

By Proposition 1, we can restrict the search to solutions of the Timed Stochastic Alignment Problem 4.3.2 of the form $(\theta^\star_1, ..., \theta^\star_n)$ where every optimal timestamp belongs to the set of observed ones. Let

$$D = \{D_0 < D_1 < \cdots < D_k\}, \textit{ with } k \leq n,$$

denote the sorted vector of distinct observed timestamps, where n is the total number of observations. By convention, we set $D_0 = 0$ to represent the initial time point. Thus the continuous Timed Stochastic Alignment Problem 4.3.2 is equivalent to a finite combinatorial problem in which, for each index i, we select a value θ_i from D. This discrete problem admits an efficient dynamic programming (DP) solution.

Given a feasible partial assignment $(\theta_1, \ldots, \theta_i)$ (with $i \leq n$), we define its *total cost* as

$$C((\theta_1, \ldots, \theta_i)) = \sum_{\ell=1}^{i} \Big(\alpha \, W_\ell \, (\theta_\ell - \theta_{\ell-1}) \; + \; \beta \, |\theta_\ell - \hat{\theta}_\ell| \Big), \tag{4.6}$$

For each candidate value $D_j \in D$ assigned to θ_i, the *value function* $V(i, j)$ is defined as the minimum total cost of a feasible partial assignment $(\theta_1, \ldots, \theta_i)$ such that $\theta_i = D_j$, namely

$$V(i, j) = \min_{\substack{(\theta_1, \ldots, \theta_i) \text{ feasible} \\ \theta_i = D_j}} C((\theta_1, \ldots, \theta_i)). \tag{4.7}$$

Given a feasible partial assignment $(\theta_1, \ldots, \theta_{i-1})$, we define the *stage cost* as the cost incurred by adding transition i, that is, by extending the assignment to $(\theta_1, \ldots, \theta_i)$ with $\theta_i = D_j$. This cost is obtained by substituting $\theta_i = D_j$ into the objective of the Timed Stochastic Alignment Problem 4.3.2:

$$\text{stageCost}(i, j) = \underbrace{\alpha \, W_i \, (\, D_j - \theta_{i-1} \,)}_{\text{likelihood cost}} \; + \; \underbrace{\beta \, |D_j - \hat{\theta}_i|}_{\text{observation cost}} . \tag{4.8}$$

The *total cost* of a feasible partial assignment $(\theta_1, \ldots, \theta_i)$ with $\theta_i = D_j$ is then given by the sum of the stage cost (4.8) and the total cost of the previous partial assignment $(\theta_1, \ldots, \theta_{i-1})$. The monotonicity constraint $\theta_{i-1} \leq \theta_i$ implies that if $\theta_i = D_j$, then θ_{i-1} must equal some D_r with $r \leq j$. Thus we can express the value function as

$$V(1, j) = \text{stageCost}(1, j), \tag{4.9}$$

$$V(i, j) = \text{stageCost}(i, j) + \min_{1 \leq r \leq j} V(i-1, r), \quad i = 2, \ldots, n, \;\; j = 1, \ldots, k. \tag{4.10}$$

The optimal final timestamp must satisfy $\theta_n \geq D_k$, hence the optimal alignment cost of a total assignment is

$$V^\star = \min_{\substack{1 \leq j \leq k \\ D_j \geq D_k}} V(n, j) = \min_{\substack{1 \leq j \leq k \\ D_j \geq D_k}} \; \min_{\substack{(\theta_1, \ldots, \theta_n) \text{ feasible} \\ \theta_n = D_j}} C((\theta_1, \ldots, \theta_n)). \tag{4.11}$$

It can be computed by dynamic programming using a table of size $n \times k$. An optimal sequence $(\theta_1^\star, \ldots, \theta_n^\star)$ is than obtained by backtracking. In particular, we

store for each (i, j) the minimizing predecessor

$$\mathrm{Ptr}(i, j) = \arg \min_{\ell : D_\ell \le D_j} \Big\{ V(i-1, \ell) + \mathrm{stageCost}(i, j \mid \ell) \Big\}.$$

Starting from

$$j^\star = \arg \min_{j : D_j \ge D_k} V(n, j), \qquad \theta_n^\star = D_{j^\star},$$

the optimal sequence is obtained recursively as

$$\theta_{i-1}^\star = D_{\mathrm{Ptr}(i, j^\star)}, \qquad i = n, n-1, \ldots, 2.$$

A naïve implementation of the recurrence (4.10) requires computing a minimum over up to j previous values, resulting in a computational complexity of $O(nk^2)$. However, the recurrence can be reformulated using *prefix minima*, reducing the runtime to $O(nk)$.

Specifically, the likelihood cost satisfies

$$V(i-1, r) + \alpha\, W_i (D_j - D_r) = V(i-1, r) - \alpha\, W_i D_r + \alpha\, W_i D_j,$$

for fixed i, only the first term depends on r. Defining the prefix minimum

$$M_j = \min_{r \le j} \Big(V(i-1, r) - \alpha\, W_i D_r \Big), \tag{4.12}$$

the dynamic programming update becomes

$$V(i, j) = M_j + \alpha\, W_i D_j + \beta\, |D_j - \hat{\theta}_i|,$$

which can be computed in $O(1)$ time per entry. Maintaining the argmin associated with each prefix minimum ensures correct backtracking, while the total complexity is reduced to $O(nk)$.

Algorithm 1 DP alignment of timestamps (naïve implementation)

Require: Observed timestamps $\theta(1{:}n)$, rates $w(1{:}n)$, weights α, β, candidate grid $D(1{:}k)$
Ensure: $\theta^\star(1{:}n)$, optimal cost
$V \leftarrow +\infty$ matrix of size $n \times k$
Ptr $\leftarrow 0$ matrix of size $n \times k$
▷ Initialization
for $j = 1$ **to** k **do**
    $V(1, j) \leftarrow \alpha w_1(D_j - 0) + \beta \, |D_j - \theta_1|$
    $\mathrm{Ptr}(1, j) \leftarrow 0$
end for
▷ DP recursion
for $i = 2$ **to** n **do**
    for $j = 1$ **to** k **do**
        $r^\star \leftarrow \arg\min_{r \le j} \Big(V(i-1, r) + \alpha w_i (D_j - D_r) \Big)$
        $V(i, j) \leftarrow V(i-1, r^\star) + \alpha w_i (D_j - D_{r^\star}) + \beta \, |D_j - \theta_i|$
        $\mathrm{Ptr}(i, j) \leftarrow r^\star$
    end for
end for
$J \leftarrow \arg\min_{j:\, D_j \ge D_k} V(n, j)$
return backtracking result and $V(n, J)$

Algorithm 2 DP alignment with prefix-minimum optimization

Require: Observed timestamps $\theta(1{:}n)$, rates $w(1{:}n)$, weights α, β, sorted candidate grid $D(1{:}k)$

Ensure: Optimal timestamps $\theta^\star(1{:}n)$, optimal cost

▷ Stage 1

for $j = 1$ **to** k **do**
  $V_{\text{prev}}(j) \leftarrow \alpha w_1 D_j + \beta |D_j - \theta_1|$
  $\text{Ptr}(1, j) \leftarrow 0$
end for

▷ Stages $i = 2, \ldots, n$

for $i = 2$ **to** n **do**
  Compute $A(j) = V_{\text{prev}}(j) - \alpha w_i D_j$
  $M(1) = A(1)$, $I(1) = 1$
  for $j = 2$ **to** k **do**
    if $A(j) < M(j-1)$ **then**
      $M(j) = A(j)$, $I(j) = j$
    else
      $M(j) = M(j-1)$, $I(j) = I(j-1)$
    end if
  end for
  for $j = 1$ **to** k **do**
    $r^\star = I(j)$
    $V_{\text{curr}}(j) = M(j) + \alpha w_i D_j + \beta |D_j - \theta_i|$
    $\text{Ptr}(i, j) = r^\star$
  end for
  $V_{\text{prev}} = V_{\text{curr}}$
end for

▷ Final state selection

$J = \arg\min_{j:\, D_j \geq D_k} V_{\text{prev}}(j)$
Backtrack using Ptr

Chapter 5

Handling Silent Transitions and Not Perfectly Fitting Models

In Chapter 4, we made strong assumptions about the alignment between the model and the observed traces: to isolate the question of timing, we focused on perfectly fitting models and assumed that all transitions were fully observable. In this chapter, we relax these assumptions and consider more realistic scenarios, where the model and the observations do not always perfectly align. We first discuss how to handle silent transitions, and then we address models that do not perfectly fit the observed traces.
Let $\hat{\sigma}$ be an observed trace. Our approach proceeds in two steps:

- **Untimed alignment:** We extract the corresponding *untimed* trace $\bar{\hat{\sigma}}$ and compute a classical alignment with the Petri net N, obtained from the stochastic Petri net N_S by disregarding temporal and frequency information. Following [10], we identify a complete run $\bar{\sigma}$ of N that minimizes the Levenshtein distance to $\hat{\sigma}$.

- **Timed alignment:** Using the trace resulting from the untimed alignment, we perform a *timed alignment* with the original stochastic Petri net N_S.

5.1 Silent Transitions

When dealing with real data log, silent transitions are particularly relevant, due to the fact that many process discovery algorithms produce as output a Petri net with silent transitions. We recall that in a *Labelled Stochastic Petri Net* N_S

there are two types of timed transitions: *visible* transitions $\lambda(t) \in \Sigma$ and *silent* transitions $\lambda(t) = \tau$. While the firing of a visible transition is explicitly recorded as an event in a trace in an event log, the firing of a silent transition instead indicates the execution of an internal step, not corresponding to any visible activity in the process.

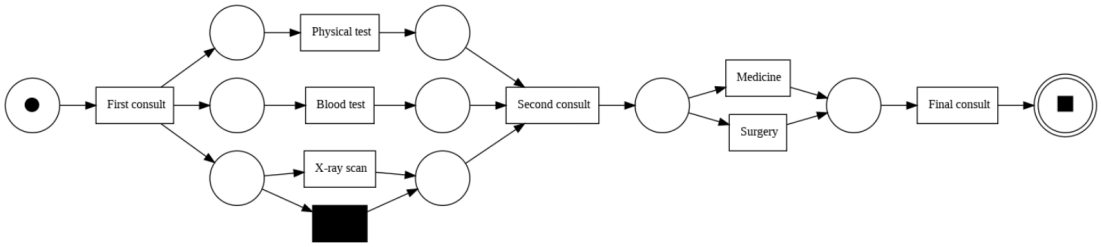

Figure 5.1: Labelled Petri Net N from the hospital dataset

As an example, we consider the labelled Petri net N shown in Figure 5.1. Our goal is to show how we handle the silent transitions in the alignment algorithm. The model N was derived from an artificial hospital event log. Firstly, we ran the Inductive Miner algorithm, introduced in [36], as implemented in the PM4Py Python framework [37] to get the untimed Petri net N depicted in Figure 5.1, then we assigned exponential firing rates in order to get a Labelled Timed Stochastic Petri Net (TSPN). For observable transition, these rates were computed based on the average firing delays observed in the log. The inductive miner introduces a silent transition in conflict with the *X-ray scan* transition in order to model the fact that this activity may or may not occur. In the literature, silent transitions have often been considered instantaneous, however in our TSPN formalism, they have firing rates like others: the idea is that silent transitions may represent internal steps, which are not observable but still take time. This idea was discussed for instance in [38]. A silent transitions can still be made almost immediate by assigning a high firing rate; however, it is well known that, in a conflict between an immediate and a timed transition, the immediate transition is always the one that fires [26]. This behaviour would not reflect the observed traces, since the *X-ray scan* activity occurs in 90% of the cases. Conversely, assigning a low firing rate to the silent transition would artificially reduce the waiting time of the subsequent *Second consult* activity, distorting its estimated rate. For these reasons, we inserted an additional silent transition before *X-ray scan*. The conflict between the two silent transitions explicitly models the decision of executing the *X-ray scan* or not. These transitions are timed but they are assumed to occur almost immediately and are therefore assigned high firing rates. In particular, a higher firing rate is assigned to the silent transition τ_1, which allows *X-ray scan*, reflecting its higher probability of execution, whereas a lower firing rate is assigned to τ_2.

The activity *First consult* plays a special role in our framework and is not treated as a regular timed transition. Since it appears in all observed traces and always represents the start of a process instance, it is used exclusively as a temporal reference point. For each trace, its timestamp is normalized to time 0, and all subsequent timestamps are expressed relative to it. As a consequence, no stochastic firing rate is associated with this transition, and it is excluded from the optimization problem. Normalizing timestamps relative to the case start and treating the initial event as a purely temporal reference is a common practice in process mining and trace analysis [39], as it removes arbitrary absolute time offsets between cases and focuses the analysis on relative temporal behavior within each trace. In our setting, excluding the initial activity from the stochastic timing model ensures that firing delays and likelihoods are comparable across traces, regardless of their absolute start times.
We rename the activities for clarity as follows:

$$\begin{aligned}
\text{Physical test} &\to \text{PT},\\
\text{Blood test} &\to \text{BT},\\
\text{X-ray scan} &\to \text{XS},\\
\text{Second consult} &\to \text{SC},\\
\text{Surgery} &\to \text{S},\\
\text{Medicine} &\to \text{M},\\
\text{Final consult} &\to \text{FC}
\end{aligned}$$

Scope and modeling assumptions. We emphasize that the objective of this thesis is *conformance checking* rather than process discovery. Consequently, any modeling choices introduced in this step—including the presence and placement of silent transitions or the assignment of firing rates—are not intended to optimize the quality of the discovered model. Instead, they serve to illustrate how the proposed alignment technique can handle realistic models that include unobservable behaviour. The corresponding labeled stochastic Petri net (TSPN) N_S is illustrated in Figure 5.2.
Now, we consider the following observed trace and show how to align it with our model N_S:

$$\begin{aligned}
\hat{\sigma} = \langle\, &(\textit{First consult}, \text{2017-01-02 11:40:11}), (\textit{Blood test}, \text{2017-01-02 12:47:33}),\\
&(\textit{Physical test}, \text{2017-01-02 12:53:50}), (\textit{Second consult}, \text{2017-01-02 16:21:06}),\\
&(\textit{Surgery}, \text{2017-01-05 13:23:09}), (\textit{Final consult}, \text{2017-01-09 08:29:28})\rangle
\end{aligned}$$

Our alignment procedure is made of two steps.

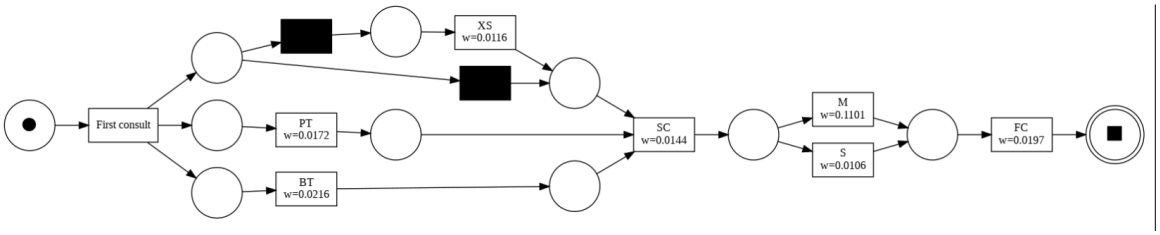

Figure 5.2: TSPN from the hospital dataset

Step 1: Untimed alignment First, we extract the untimed version of the trace by removing the temporal information:

$$\begin{aligned}\bar{\hat{\sigma}} =&\langle \textit{First consult}, \textit{Blood test}, \textit{Physical test},\\ &\textit{Second consult}, \textit{Surgery}, \textit{Final consult}\rangle\\ =&\langle \textit{First consult}, BT, PT, SC, S, FC\rangle\end{aligned}$$

We then compute the classical alignment between this untimed trace and the Petri net N obtained from N_S disregarding stochastic and temporal informations.

$$\begin{array}{rccccccc}\text{log:} & \textit{First consult} & \gg & BT & PT & SC & S & FC\\ \text{model:} & \textit{First consult} & \tau_2 & BT & PT & SC & S & FC\end{array}$$

where $\binom{\gg}{\tau_2}$ denotes a silent model move (insertion), with cost 0. The optimal alignment requires the insertion of the silent transition τ_2 after the activity *First consult* and before the activity *Second consult*. Since this move has zero cost, the resulting alignment is perfect. The untimed alignment yields the model trace

$$\bar{\sigma'} = \langle \textit{First consult}, \tau_2, BT, PT, SC, S, FC, \rangle$$

Thus in this case we still assume the model to be perfectly fitting, in the sense that, for every observed trace $\hat{\sigma}$ that we consider, the model has a run σ' such that $\lambda(\bar{\sigma'}) = \bar{\hat{\sigma}}$. Simply, we now allow models which have silent transitions, which means that σ' may be only partially observed.

Step 2: Timed alignment To perform the timed alignment, we construct a corresponding timed trace by assigning observed timestamps only to the transitions that are actually observed. The timestamps are expressed in hours and relatively to the *First consult*, which is excluded from the optimization, since it is common to all traces and serves only as the initial reference point. The resulting (partially) timed trace is therefore

$$\sigma' = \langle (\tau_2, -), (BT, 1.1228), (PT, 1.2275), (SC, 4.6819), (S, 73.7161), (FC, 164.8214)\rangle$$

Let $\mathcal{O} \subseteq \{1, \ldots, n\}$ denote the set of indices of transitions in σ' with observed timestamps, in order to allow silent transitions to influence the likelihood of the execution without contributing to the timestamp deviation term, we modify the Timed Stochastic Alignment Problem 4.3.2 as follows.

$$\begin{aligned} \min_{\theta_1,\ldots,\theta_n} \quad & \alpha \sum_{i=1}^{n} W_{t_i}(\theta_i - \theta_{i-1}) + \beta \sum_{i \in \mathcal{O}} \left|\theta_i - \hat{\theta}_i\right| \\ \text{subject to} \quad & \theta_i \leq \theta_{i+1}, \quad \forall i = 1, \ldots, n-1 \\ & \theta_n \geq \max_{i \in \mathcal{O}} \hat{\theta}_i \end{aligned} \tag{5.1}$$

This ensures that only observed transitions are penalized for deviation from the observed timestamps, while all transitions, including silent ones, are considered in the likelihood component of the objective. The last constraint enforces that the final timestamp θ_n is at least as large as the biggest observed timestamp and can be linearized as the set of constraints

$$\theta_n \geq \hat{\theta}_i \quad \forall\, i \in \mathcal{O}$$

Thus the problem remains linear and Proposition 1 still holds. In the DP approach, we adjust the *stage cost* to account for transitions with missing observed timestamps. Formally, the modified stage cost is

$$\text{stageCost}(i, j) = \alpha\, W_i\, (D_j - \theta_{i-1}) \; + \; \begin{cases} \beta\, |D_j - \hat{\theta}_i|, & \hat{\theta}_i \text{ observed}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

This reformulation allows silent transitions to contribute to the likelihood term while not affecting the timestamp deviation.

5.2 Not-perfectly aligned untimed traces

So far, we have restricted our attention to settings in which the model can perfectly replay the observed trace. We now extend our approach to more realistic scenarios, where the observed traces cannot be replayed exactly by the model. We show that the approach used in the case where the only inserted transitions were the silent ones still applies in this setting. In particular, we consider the model shown in Figure 4.1 and assume that the following timed trace is observed:

$$\hat{\sigma} = \langle (a,10.2), (b,11.3), (d,14.1), (c,14.9) \rangle.$$

Step 1: Untimed alignment We first extract the untimed trace $\bar{\hat{\sigma}} = \langle a, b, d, c \rangle$. The optimal untimed alignment consists in swapping the activities c and d. Operationally, this is achieved by deleting d and inserting it after c, resulting in a total alignment cost of 2.

$$\begin{array}{lccccc} \text{log:} & a & b & d & c & \gg \\ \text{model:} & a & b & \gg & c & d \end{array}$$

where $\binom{d}{\gg}$ denotes a log move (deletion), while $\binom{\gg}{d}$ denotes a model move (insertion), each with unit cost.

Step 2: Timed alignment The untimed alignment yields the following model trace: $\bar{\sigma}' = \langle a, b, c, d \rangle$. To perform the timed alignment, we construct a corresponding timed trace as in Section 5.1, assigning the observed timestamps only to transitions that already align with the model, while inserted transitions (model moves) are considered as unobserved transitions and treated like silent transitions. The resulting (partially) timed trace is therefore

$$\sigma' = \langle (a,10.2), (b,11.3), (c,14.9), (d, -) \rangle,$$

and the timed alignment can be performed using the optimization problem formulation (5.1).

5.3 Limitations of the two-step alignment approach

While the proposed two-step alignment approach effectively handles silent transitions and non-perfect model fits, it relies on a strict separation between structural and temporal optimization. Specifically, the untimed alignment is computed without considering timestamps, and the timed alignment is applied only after the model trace has been fixed. As a result, temporal information cannot influence the choice of structural deviations, which may lead to suboptimal alignments when timestamps carry strong evidence. Inserted model moves are treated like silent transitions at the timed level. Their timestamp deviations are not penalized, which allows the firing rates of these transitions to affect the likelihood correctly; otherwise, enforcing a temporal penalty could change the order of transitions, thereby altering the rates and distorting the stochastic behaviour of the model.
In the example discussed in Section 5.2, the observed timestamps clearly indicate that the activity d occurs before c ($d \prec c$). However, the untimed alignment enforces the model order $c \prec d$, which cannot be violated. As a result, the timed alignment must artificially delay d to occur after c. This prevents the timestamp deviation for d from being minimized, since it can no longer match its observed

occurrence, while allowing the likelihood to be maximized under the assumption that d happened after c. Consequently, although the alignment is optimal with respect to the untimed structural objective, it is suboptimal when considering the combined structural and temporal objective.
This approach represents a first attempt to address the problem. A natural extension would be a unified alignment framework jointly optimizing structural, temporal, and stochastic costs.

Chapter 6

Aligning Partial-Order Traces

In the previous Chapters 4 and 5, we performed the alignment of observed log traces to a model assuming a total order of events. However, in many systems, some transitions are concurrent and can occur in any order without affecting the overall system behavior. The idea of incorporating partial orders into the definition of alignments was first proposed in [40]. By considering *partial orders* in alignments, we can capture this concurrency and provide more plausible explanations for observed traces.
Consider again the motivating example of Figure 4.1, and focus on the log trace

$$\langle(a,12.3),(c,12.5),(b,13.2),(d,19.1)\rangle.$$

Here, action c appears before b, even though c typically has a longer expected waiting time than b (rate 1 for b and 0.2 for c). Moreover, the waiting time before d (5.9 time units) is unusually long. A plausible explanation is that c actually fired later but was recorded incorrectly. Solving the Timed Stochastic Alignment Problem 4.3.2 of Chapter 4, the best achievable alignment is:

$$\langle(a,12.3),(c,\theta),(b,\theta),(d,19.1)\rangle,$$

where c fires immediately before b at some $\theta \in [12.5,19.1]$. While this alignment is slightly more likely than the observed trace, it does not fully capture the most plausible system behaviour. This limitation arises because delaying c changes the order of b and c, which our previous alignment method cannot handle. To address this limitation, we adapt the definition of alignments to allow permutations of concurrent actions. Following the notion of Mazurkiewicz traces [41], two untimed runs σ and σ' of a TSPN N_S are considered equivalent if they differ only in the order of transitions that are concurrent according to the model. In our example, b

and c are concurrent, since their presets do not intersect. This places $\langle a, b, c, d\rangle$ and $\langle a, c, b, d\rangle$ in the same equivalence class. Adopting this partial-order perspective allows alignments to better capture the underlying system dynamics. Now our alignment problem can be formalized as follows.

Definition 6.0.1 (Partial-Order Alignment of a Timed Trace to a TSPN). Given an observed trace

$$\hat{\sigma} = \langle (a_1, \hat{\theta}_1), \ldots, (a_n, \hat{\theta}_n)\rangle$$

and a TSPN

$$N_S = (P, T, F, \Sigma, \lambda, w, M_0),$$

a *partial-order alignment* of $\hat{\sigma}$ to N_S is a run

$$\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n)\rangle$$

of N_S such that $\bar{\sigma}$ and $\bar{\hat{\sigma}}$ are equivalent in the sense of Mazurkiewicz traces, and the firing times $\theta_1, \ldots, \theta_n$ minimize the objective function

$$\alpha\,(-\ell(\sigma)) \;+\; \beta\,\|\pi(\boldsymbol{\theta}) - \hat{\boldsymbol{\theta}}\|_{L_1}, \tag{6.1}$$

where $\ell(\sigma)$ denotes the likelihood of σ, $\alpha, \beta \geq 0$ are tunable parameters such that $\alpha + \beta = 1$, and π denotes the permutation for which $\pi(\bar{\sigma}) = \bar{\hat{\sigma}}$; the firing times must be permuted accordingly.

In essence, the partial-order alignment finds a model run consistent with the observed trace up to concurrent transition order. The objective function (6.1) balances model likelihood with temporal deviation from observations. The permutation π ensures that firing times are compared in a way that respects the equivalence of untimed traces under concurrent transitions. By adjusting the parameters α and β, one can give more weight either to the likelihood of the run or to the temporal distance to the observed trace.
Coming back to our example of the observed trace

$$\langle (a,12.3), (c,12.5), (b,13.2), (d,19.1)\rangle,$$

the partial-order alignment problem now allows to choose the best run of the model among the runs of the form

$$\langle (a, \theta_a), (c, \theta_c), (b, \theta_b), (d, \theta_d)\rangle \quad \text{and} \quad \langle (a, \theta_a), (b, \theta_b), (c, \theta_c), (d, \theta_d)\rangle,$$

because their untimed supports, $\langle a, c, b, d\rangle$ and $\langle a, b, c, d\rangle$, are equivalent in the sense of Mazurkiewicz traces. We solve the Timed Stochastic Alignment Problem 4.3.2 for the two possible orders using the method introduced in Chapter 4. For

example, if we consider $\alpha = 0.7$, for the first trace (which preserves the observed order) we obtain:

$$\langle (a,13.2), (c,13.2), (b,13.2), (d,19.1) \rangle$$

with a corresponding objective function value of 5.4759. For the second trace (in which we do not fully trust the observation and consider the most likely order of transitions) we obtain:

$$\langle (a,13.2), (b,13.2), (c,19.1), (d,19.1) \rangle,$$

with a corresponding objective function value of 4.7235. Since this value is lower, the solution has improved, which aligns with our intuition that this is a more plausible explanation of the observation when the likelihood of the aligned trace is taken into account.
A straightforward approach to the partial-order alignment problem is to evaluate each equivalent trace by considering all permutations of concurrent transitions. However, this would require solving as many optimization problems as there are equivalent Mazurkiewicz traces and then comparing the outcomes.

6.1 Likelihood of Partial-Order Traces

In this section, we show that the objective function for the alignment can be expressed in a form that is common to all equivalent traces. This allows one to find the partial-order alignment by solving a single optimization problem for this common formula. To achieve this, we focus on a popular class of Petri nets known as *extended free-choice Petri nets*, introduced in Chapter 2, Section 2.2.4.
For example, in Figure 4.1, consider transition c. Its conflict cluster K_c is formed by c itself and the transition c' with which it is in conflict:

$$K_c = \{c, c'\}.$$

For extended free-choice Petri nets, the likelihood formula can be expressed without explicitly considering the order of concurrent transitions. To formalize this idea, we first recall some definitions, then present a concrete example, and finally state a formal lemma.

Definition 6.1.1 (Enabling Time). Let

$$\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n) \rangle$$

be a run of a timed stochastic Petri net $N_S = (P, T, F, \Sigma, \lambda, w, M_0)$.
Denote by $M_1, \ldots, M_n$ the markings reached along this run. For simplicity, assume that all transitions t_i are distinct, so we can refer to the firing time of a transition

without mentioning its position in the trace: $\theta_t = \theta_i$ when $t = t_i$. The *enabling time* of transition t_i is the earliest time at which t_i became enabled:

$$e_{t_i} = \min\{\theta_j \mid j \in \{0, \ldots, i-1\},\ t_j \text{ is enabled in marking } M_j\}.$$

Returning to Figure 4.1, consider the timed trace

$$\sigma = \langle (a,12.3), (c,12.5), (b,13.2), (d,19.1) \rangle$$

introduced at the beginning of Chapter 6. The corresponding enabling times are

$$\boldsymbol{e} = (e_a = 0,\ e_c = 12.3,\ e_b = 12.3,\ e_d = 13.2).$$

In particular, the enabling time of transition d is

$$e_d = \max(\theta_b, \theta_c) = 13.2.$$

An important observation is that the enabling time of a transition t can be expressed in terms of its causal predecessors:

$$^{\bullet\bullet}t = \{t' \mid t'^{\bullet} \cap {}^{\bullet}t \neq \emptyset\}.$$

Crucially, these causal predecessors do not depend on the ordering of concurrent transitions.
Note that in extended free-choice Petri nets, enabling times depend only on the causal structure of the net, and not on the particular linearization of concurrent transitions. In other words, if two runs differ only by a permutation of concurrent transitions (i.e., they are Mazurkiewicz-equivalent), each transition has the same enabling time in both runs. This invariance of enabling times under concurrent permutations is the key property established in Lemma 6.1.1.
Before formally stating the lemma, we illustrate this property with a simple example to provide intuition.

Example (Likelihood of Partial-Order Traces).
Consider again the motivating example and the two untimed traces

$$\bar{\sigma}_1 = \langle a, b, c', d \rangle, \quad \bar{\sigma}_2 = \langle a, c', b, d \rangle.$$

These traces are equivalent in the sense of Mazurkiewicz traces, as they differ only by the permutation of transitions b and c', which are concurrent. The corresponding timed traces are

$$\sigma_1 = \langle (a,\theta_1), (b,\theta_2), (c',\theta_3), (d,\theta_4) \rangle, \quad \sigma_2 = \langle (a,\theta_1), (c',\theta_2), (b,\theta_3), (d,\theta_4) \rangle.$$

Using the standard TSPN likelihood formula (4.1), we have:

$$
\begin{aligned}
\ell(\sigma_1) &= w_a e^{-w_a\theta_1} \cdot w_b e^{-(w_b+w_c+w_{c'})(\theta_2-\theta_1)} \cdot w_{c'} e^{-(w_c+w_{c'})(\theta_3-\theta_2)} \cdot w_d e^{-w_d(\theta_4-\theta_3)} \\
&= w_a e^{-w_a\theta_1} \cdot w_b e^{-(w_b+\Gamma_{c'})(\theta_2-\theta_1)} \cdot w_{c'} e^{-\Gamma_{c'}(\theta_3-\theta_2)} \cdot w_d e^{-w_d(\theta_4-\theta_3)} \\
&= w_a e^{-w_a\theta_1} \cdot w_b e^{-w_b(\theta_2-\theta_1)} \cdot w_{c'} e^{-\Gamma_{c'}(\theta_3-\theta_1)} \cdot w_d e^{-w_d(\theta_4-\theta_3)}, \\
\ell(\sigma_2) &= w_a e^{-w_a\theta_1} \cdot w_{c'} e^{-(w_b+w_c+w_{c'})(\theta_2-\theta_1)} \cdot w_b e^{-w_b(\theta_3-\theta_2)} \cdot w_d e^{-w_d(\theta_4-\theta_3)} \\
&= w_a e^{-w_a\theta_1} \cdot w_{c'} e^{-(w_b+\Gamma_{c'})(\theta_2-\theta_1)} \cdot w_b e^{-w_b(\theta_3-\theta_2)} \cdot w_d e^{-w_d(\theta_4-\theta_3)} \\
&= w_a e^{-w_a\theta_1} \cdot w_{c'} e^{-\Gamma_{c'}(\theta_2-\theta_1)} \cdot w_b e^{-w_b(\theta_3-\theta_1)} \cdot w_d e^{-w_d(\theta_4-\theta_3)},
\end{aligned}
$$

where $\Gamma_{c'} = w_c + w_{c'}$. Using the enabling times

$$e_a = \theta_0 = 0, \quad e_b = e_{c'} = \theta_1, \quad e_d = \theta_3,$$

we can rewrite the likelihoods as

$$\ell(\sigma_1) = w_a e^{-w_a(\theta_a-e_a)} \cdot w_b e^{-w_b(\theta_b-e_b)} \cdot w_{c'} e^{-\Gamma_{c'}(\theta_{c'}-e_{c'})} \cdot w_d e^{-w_d(\theta_d-e_d)}, \tag{6.2}$$

$$\ell(\sigma_2) = w_a e^{-w_a(\theta_a-e_a)} \cdot w_{c'} e^{-\Gamma_{c'}(\theta_{c'}-e_{c'})} \cdot w_b e^{-w_b(\theta_b-e_b)} \cdot w_d e^{-w_d(\theta_d-e_d)}. \tag{6.3}$$

A direct comparison shows that $\ell(\sigma_1) = \ell(\sigma_2)$, confirming that the likelihood depends only on enabling times and conflict clusters, not on the order of concurrent transitions.

Lemma 6.1.1. *For extended free-choice Petri nets, the likelihood of a timed trace $\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n) \rangle$ can be expressed as*

$$\ell(\sigma) = \prod_{i=1}^{n} w_{t_i} \, e^{-\Gamma_{t_i}(\theta_{t_i}-e_{t_i})},$$

where $\Gamma_{t_i} = \sum_{k \in K_{t_i}} w_{t_k}$ and K_{t_i} is the conflict cluster of t_i.

Proof. Recall that the standard SPN likelihood for a timed trace $\sigma = \langle (t_1, \theta_1), \ldots, (t_n, \theta_n) \rangle$ with $\theta_0 = 0$ is

$$\ell(\sigma) = \prod_{i=1}^{n} w_{t_i} \, e^{-W_{t_i}(\theta_i-\theta_{i-1})}.$$

Let w_σ be the set of transitions that actually fire in σ. We want to rewrite this expression in a form that factors over conflict clusters. Let T be the set of all transitions, $\bar{\sigma} = \langle t_1 \ldots t_n \rangle$ the untimed trace, and w_σ the set of transitions that actually fire in σ.

$$
\begin{aligned}
\ell(\sigma) &= \prod_{i=1}^{n} w_{t_i}\, e^{-W_{t_i}(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{i=1}^{n} e^{-W_{t_i}(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{i=1}^{n} e^{-\sum_{t\in en(M_i)} w_t(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{i=1}^{n} \prod_{t\in en(M_i)} e^{-w_t(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t\in T} \prod_{\substack{i \\ t\in en(M_i)}} e^{-w_t(\theta_i-\theta_{i-1})} \\
&\quad \text{every transition either fires or is in conflict with a firing transition} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t'\in w_\sigma} \prod_{t\in K_{t'}} \prod_{\substack{i \\ t\in en(M_i)}} e^{-w_t(\theta_i-\theta_{i-1})} \\
&\quad \text{in extended free-choice SPNs, } t \in en(M_i) \iff t' \in en(M_i) \text{ for } t \in K_{t'} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t'\in w_\sigma} \prod_{t\in K_{t'}} \prod_{\substack{i \\ t'\in en(M_i)}} e^{-w_{t'}(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t'\in w_\sigma} \prod_{\substack{i \\ t'\in en(M_i)}} \prod_{t\in K_{t'}} e^{-w_{t'}(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t'\in w_\sigma} \prod_{\substack{i \\ t'\in en(M_i)}} e^{-\Gamma_{t'}(\theta_i-\theta_{i-1})} \\
&= \prod_{i=1}^{n} w_{t_i} \prod_{t'\in w_\sigma} e^{-\Gamma_{t'}(\theta_{t'}-e_{t'})} \\
&= \prod_{i=1}^{n} w_{t_i}\, e^{-\Gamma_{t_i}(\theta_{t_i}-e_{t_i})}.
\end{aligned}
$$

Thus, in an extended free-choice SPN, the contribution of a firing transition $t' \in w_\sigma$ and its conflict cluster $K_{t'}$ can be factored as

$$
\prod_{\substack{i \\ t'\in en(M_i)}} \prod_{t\in K_{t'}} e^{-w_t(\theta_i-\theta_{i-1})} = \prod_{\substack{i \\ t'\in en(M_i)}} e^{-\Gamma_{t'}(\theta_i-\theta_{i-1})}.
$$

Collecting these terms over all firing transitions and using the enabling times $e_{t'}$

yields

$$\ell(\sigma) = \prod_{i=1}^{n} w_{t_i}\, e^{-\Gamma_{t_i}(\theta_{t_i} - e_{t_i})},$$

which proves the lemma. □

6.2 Optimization Problem Formulation

In the general case, solving the partial-order alignment problem corresponds to solving the Timed Stochastic Alignment Problem 4.3.2 for *all possible linearizations* of the partial order. The number of linearizations grows exponentially with the degree of concurrency, making this approach computationally challenging.
Using Lemma 6.1.1, we can adapt the Timed Stochastic Alignment Problem 4.3.2 introduced in Chapter 4 to the setting of partial-order traces.

From the likelihood expression obtained in Lemma 6.1.1, the log-likelihood of a trace σ is

$$\mathcal{L}(\sigma) = \log(\ell(\sigma)) = \sum_{i=1}^{n} \log(w_{t_i}) \; - \; \Gamma_{t_i}\,(\theta_{t_i} - e_{t_i}). \tag{6.4}$$

As in the fixed-order case, the terms $\log(w_{t_i})$ are constant with respect to θ and can be omitted in the optimization. The timestamp-based distance term remains unchanged. The key difference in the partial-order setting is that the enabling times e_{t_i} depend on multiple possible predecessors due to concurrency. Therefore, we must introduce constraints that encode all admissible linearizations consistent with the partial order. The resulting optimization problem can be written as follows.

Definition 6.2.1 (Partial Order Timed Stochastic Alignment Problem).

$$\begin{aligned}
\min_{\boldsymbol{\theta},\,\mathbf{e}} \quad & \alpha \sum_{i=1}^{n} \Gamma_{t_i}\,(\theta_{t_i} - e_{t_i}) \; + \; \beta \sum_{i=1}^{n} \left|\theta_{t_i} - \hat{\theta}_{t_i}\right| \\
\text{s.t.} \quad & e_{t_i} = \max_{t_j \in {}^{\bullet\bullet}t_i} \theta_{t_j}, && \forall\, i = 1, \ldots, n, \\
& \theta_{t_i} \geq e_{t_i}, && \forall\, i = 1, \ldots, n, \\
& \theta_{t_n} \geq \hat{\theta}_{t_n}.
\end{aligned} \tag{6.5}$$

Here, the optimization involves two groups of variables: $\boldsymbol{\theta}$ and $\mathbf{e}$.
The max-operator in the enabling-time constraints introduces nonlinearity, and cannot be linearized with inequalities of the form

$$e_{t_i} \geq \theta_{t_j}, \quad \forall t_j \in {}^{\bullet\bullet}t_i.$$

This is because, in maximizing the likelihood, the objective effectively *minimizes the term* $(\theta_{t_i} - e_{t_i})$, which is multiplied by α. As a result, an optimizer would tend to choose e_{t_i} as large as possible, which is not the behaviour we want. Consequently, classical linear programming solvers are not directly applicable.

An alternative approach is to enumerate all *linearizations* (i.e., topological orders) consistent with the partial order. For each fixed linearization, the optimization problem reduces to a convex program: the enabling-time constraints become simple linear inequalities without the max operator.
Solving these convex programs individually and selecting the best solution over all linearizations yields a global optimum. However, this approach is only computationally feasible for small numbers of concurrent events, as the number of linearizations grows exponentially with the degree of concurrency in the partial order.

6.3 Convexity and Uniqueness of Solutions

The max-operator in the enabling-time constraints introduces nonlinearity. As in the simpler case presented in Chapter 4, the objective function

$$\alpha \sum_{i=1}^{n} \Gamma_{t_i}(\theta_{t_i} - e_{t_i}) + \beta \sum_{i=1}^{n} |\theta_{t_i} - \hat{\theta}_{t_i}|$$

is convex in $(\boldsymbol{\theta}, \mathbf{e})$. Indeed, the terms $\Gamma_{t_i}(\theta_{t_i} - e_{t_i})$ are linear, and the absolute-value terms $|\theta_{t_i} - \hat{\theta}_{t_i}|$ are convex. The linear constraints $\theta_{t_i} \geq e_{t_i}$ and $\theta_{t_n} \geq \hat{\theta}_{t_n}$ define a convex feasible region. The equality constraints

$$e_{t_i} = \max_{t_j \in {}^{\bullet\bullet}t_i} \theta_{t_j}$$

express each e_{t_i} as the pointwise maximum of a set of affine functions of $\boldsymbol{\theta}$. Since the maximum of convex functions is convex, these constraints preserve the convexity of the feasible set, even though they are nonlinear.
Overall, the optimization problem is convex. However, it is *not strictly convex*: the objective contains no strictly convex term, and the feasible set may contain flat directions induced by the max-operator. As a result, the problem does not admit a uniqueness guarantee and may have multiple optimal solutions.
Example
Consider a simplified version of the model in Figure 4.1, where the conflict between c and c' is removed and the rates are $w_a = w_b = w_c = 1$, $w_d = 2$.
Suppose we observe the trace

$$\hat{\sigma} = \langle (a,1), (c,2), (b,2), (d,15) \rangle.$$

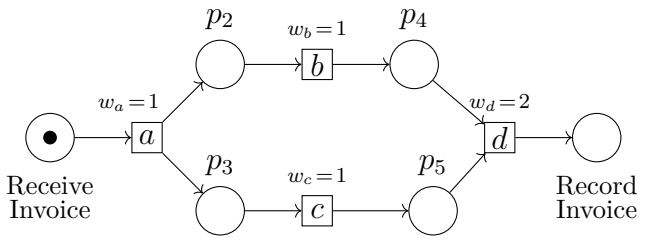

Figure 6.1: Timed stochastic Petri net model.

It may seem surprising that d occurs much later than b and c, which have the same timestamp. One may wonder whether the relative order of b and c affects the enabling time of d.
Let θ_b and θ_c denote the firing times of b and c. Transition d becomes enabled only after receiving tokens from both input places p_4 and p_5:

$$e_d = \max\{\theta_b, \theta_c\}.$$

No matter which of b or c fires first, the enabling time of d is determined by the later of the two firing times:

- If b fires first ($\theta_b < \theta_c$), then d is enabled only after c fires, so $e_d = \theta_c$.
- If c fires first ($\theta_c < \theta_b$), then $e_d = \theta_b$.

Hence, the relative order of b and c does not affect e_d. Consequently, any permutation of b and c at the same timestamp yields the same enabling time and the same contribution to the likelihood term. Since b and c also have the same firing rate, and in the observed trace they occur at the same timestamp, their contribution to the timestamp distance term is symmetric.

Let $\boldsymbol{\theta}^{(1)}$ be an optimizer with b "before" c, i.e., $\theta_b^{(1)} \leq \theta_c^{(1)}$. Define $\boldsymbol{\theta}^{(2)}$ by swapping these two entries:

$$\theta_b^{(2)} = \theta_c^{(1)}, \quad \theta_c^{(2)} = \theta_b^{(1)},$$

and leaving all other variables unchanged.

- All constraints remain satisfied because the max operation in $e_d = \max\{\theta_b, \theta_c\}$ is symmetric.
- The objective value remains exactly the same due to symmetry in both the likelihood and the timestamp distance terms.

Hence, $\boldsymbol{\theta}^{(2)}$ is also optimal.
This shows that the optimization problem admits at least two distinct optima, corresponding to the two possible orders of b and c. In general, the solution is *not unique*.

6.4 NP-Hardness of the Partial-Order Alignment Problem

In [42, 43] Pinedo defined a *scheduling problem* as an optimization problem in which a set of activities (called *jobs*) must be assigned to limited resources over time, subject to constraints, in order to optimize a given objective function.
In this section we show that the Partial Order Timed Stochastic Alignment Problem (6.2.1) is NP-hard by means of a polynomial-time reduction from the *single-machine scheduling problem* $1|\text{prec}|\sum_j w_j C_j$, which is a scheduling problem with precedence constraints on a single machine with the goal to minimize the weighted sum of completion times. This problem is known to be strongly NP-hard [44, 42, 43].

Source Problem: $1|\text{prec}|\sum_j w_j C_j$
An instance of the single-machine scheduling problem with precedence constraints and weighted completion time objective is specified by:

$$J = \{1, \ldots, n\}, \quad p_j > 0, \quad w_j > 0, \quad \text{and a precedence relation } \prec \text{ on } J,$$

where p_j is the processing time and w_j the weight of job j. We recall that, as defined in [43], a *precedence relation* R is a partial order on J. If a job i must be completed before another job j may start, then $(i, j) \in R$. In this case, j is a (direct) successor of i, and i is a (direct) predecessor of j. We consider *non-preemptive* schedules: once the machine starts processing a job, it executes it without interruption until completion.
A non-preemptive schedule on a single machine corresponds to a linear extension π of the partial order $\prec$: π is a permutation of $J = \{1, \ldots, n\}$ such that if $k \prec j$, then $\pi(k) < \pi(j)$. Jobs are executed in the order $\pi(1), \pi(2), \ldots, \pi(n)$ without overlap. The completion times are defined recursively as:

$$C_{\pi(1)} = p_{\pi(1)}, \quad C_{\pi(k)} = C_{\pi(k-1)} + p_{\pi(k)}, \quad k = 2, \ldots, n.$$

The completion time of job j in schedule π is denoted by $C_j(\pi)$. The objective is to minimize the weighted sum of completion times:

$$\sum_{j \in J} w_j C_j(\pi).$$

The corresponding decision problem asks whether there exists a feasible schedule π such that

$$\sum_{j=1}^{n} w_j C_j(\pi) \leq K,$$

for a given bound K. This decision problem is NP-complete [43].

Target Problem: Partial-Order Alignment
We consider a restricted instance of (6.5) with $\alpha = 1$, $\beta = 0$, and without the terminal constraint. The optimization problem reduces to:

Definition 6.4.1 (Restricted Partial-Order Alignment Problem).

$$\begin{aligned} \min_{\boldsymbol{\theta},\mathbf{e}} \quad & \sum_{i=1}^{n} \Gamma_{t_i}(\theta_{t_i} - e_{t_i}) \\ \text{s.t.} \quad & e_{t_i} = \max_{t_j \in {}^{\bullet\bullet}t_i} \theta_{t_j}, \quad i = 1, \ldots, n, \\ & \theta_{t_i} \geq e_{t_i}, \quad i = 1, \ldots, n. \end{aligned} \tag{6.6}$$

Since (6.6) is a special case of (6.5), proving NP-hardness for this restricted problem suffices.

Reduction from Scheduling
Given an instance of $1|\text{prec}|\sum_j w_j C_j$, we construct an instance of the Restricted Partial-Order Alignment Problem 6.4.1 as follows:

- For each job $j \in J$, introduce a transition t_j.
- Set the conflict cluster weight $\Gamma_{t_j} := w_j$.
- Define the partial order on transitions to match the job precedence: $t_k \in {}^{\bullet\bullet}t_j \iff k \prec j$.
- Introduce variables θ_{t_j} (completion time) and e_{t_j} (enabling time).
- Encode the processing times as linear constraints:

$$\theta_{t_j} - e_{t_j} = p_j, \quad \forall j \in J.$$

- Enforce precedence constraints with

$$e_{t_j} = \max_{k \prec j} \theta_{t_k}, \quad \forall j \in J,$$

  with $e_{t_j} = 0$ for jobs without predecessors.

Constraints on θ and e ensure that θ_{t_j} equals the completion time of job j in a non-preemptive schedule respecting $\prec$. Thus, the mapping

$$\pi \longmapsto (\theta_{t_j} = C_j(\pi),\ e_{t_j} = C_j(\pi) - p_j)$$

defines a bijection between schedules π and feasible solutions of the alignment problem. The objective functions are identical:

$$\sum_j w_j \theta_{t_j} = \sum_j w_j C_j(\pi).$$

Hence, the decision version of the Restricted Partial-Order Alignment Problem 6.4.1 is NP-complete, implying that the partial-order alignment problem is NP-hard.

Case $\beta = 1$
When $\beta = 1$ and $\alpha = 0$, the optimization problem

$$\begin{aligned} \min_{\boldsymbol{\theta},\mathbf{e}} \quad & \sum_{i=1}^{n} \left|\theta_{t_i} - \hat{\theta}_{t_i}\right| \\ \text{s.t.} \quad & e_{t_i} = \max_{t_j \in {}^{\bullet\bullet}t_i} \theta_{t_j}, \quad i = 1, \ldots, n, \\ & \theta_{t_i} \geq e_{t_i}, \quad i = 1, \ldots, n, \\ & \theta_{t_n} \geq \hat{\theta}_{t_n}, \end{aligned} \tag{6.7}$$

can be reformulated as a linear program. Indeed, the equality

$$e_{t_i} = \max_{t_j \in {}^{\bullet\bullet}t_i} \theta_{t_j}$$

together with the constraint $\theta_{t_i} \geq e_{t_i}$ is equivalent to imposing the precedence constraints

$$\theta_{t_i} \geq \theta_{t_j}, \quad \forall t_j \in {}^{\bullet\bullet}t_i,\ i = 1, \ldots, n.$$

Hence the problem can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{i=1}^{n} \left|\theta_{t_i} - \hat{\theta}_{t_i}\right| \\ \text{s.t.} \quad & \theta_{t_i} \geq \theta_{t_j}, \quad \forall t_j \in {}^{\bullet\bullet}t_i,\ i = 1, \ldots, n, \\ & \theta_{t_n} \geq \hat{\theta}_{t_n}. \end{aligned} \tag{6.8}$$

The absolute values in the objective function can then be linearized as in the Timed Stochastic Alignment Problem 4.3.2 by introducing auxiliary variables $u_i \geq 0$ such that

$$u_i \geq \theta_{t_i} - \hat{\theta}_{t_i}, \qquad u_i \geq -(\theta_{t_i} - \hat{\theta}_{t_i}),$$

and minimizing $\sum_{i=1}^{n} u_i$. All constraints are now linear, yielding a linear programming formulation.
Thus, this special case is solvable in polynomial time.

6.5 Solution Approaches

In this section, we discuss different algorithmic strategies for solving the Partial Order Timed Stochastic Alignment Problem 6.2.1.

6.5.1 Non-linear solver

A direct approach to solving the Partial Order Timed Stochastic Alignment Problem 6.2.1 would be to employ a *nonlinear solver*, which can handle the nonlinear constraints introduced by the maximum operator. However, such solvers generally provide guarantees only for convergence to *local* optima, as different initializations may lead to different solutions.

Algorithm 3 Nonlinear partial-order alignment via constrained optimization

Require: rates $\Gamma(1{:}n)$, initial timestamps $\hat{\theta}(1{:}n)$, predecessor lists, weights α, β
Ensure: Optimal timestamps $\theta^\star(1{:}n)$ and auxiliary variables $e^\star(1{:}n)$
1: Construct initial guess $x_0 = [\hat{\theta}; e_0]$ using max of predecessors
2: Define variable bounds, linear constraints, and nonlinear constraints enforcing partial order
3: Define objective $f(x) = \alpha \sum_i \Gamma_i(\theta_i - e_i) + \beta \sum_i |\theta_i - \hat{\theta}_\mathrm{i}|$
4: Solve nonlinear program using `fmincon`
5: Extract $\theta^\star$ and $e^\star$ from solution

In contrast, our goal is to develop methods that guarantee convergence to a *global* optimum, building on the techniques introduced in Chapters 4 and 5.
In the fixed-order case, the optimization problem admits an exact solution via *dynamic programming (DP)*. In the partial-order setting, however, enabling times are no longer uniquely determined by a single predecessor, since concurrency allows for multiple admissible linearizations of the event set.
Each topological order induces a different convex optimization problem, and the number of such orders grows exponentially with the degree of concurrency.
Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ denote a topological order consistent with the precedence constraints, and define the reordered variables

$$\theta_i := \theta_{\pi_i}, \qquad \hat{\theta}_i := \hat{\theta}_{\pi_i}.$$

This observation naturally leads to a two-level decomposition:

1. selection of a topological order $\boldsymbol{\pi}$;
2. optimization of timestamps $\boldsymbol{\theta}$ for a fixed $\boldsymbol{\pi}$.

For a fixed ordering $\boldsymbol{\pi}$, the Partial Order Timed Stochastic Alignment Problem 6.2.1 reduces to the Timed Stochastic Alignment Problem 4.3.2 and can be solved exactly using dynamic programming.
In order to apply the DP algorithm introduced in Chapter 4, Section 4.4, for a given topological order, we explicitly compute the corresponding order-dependent transition rates. This preprocessing step requires only a single linear pass over the

ordered transitions and has minimal cost compared to the DP itself.
Although it would be possible to derive an alternative DP formulation directly based on enabling times and predecessor relations, incorporating enabling-time constraints inside the DP recursion would require reconstructing predecessor timestamps at each DP state. This would significantly increase the complexity of the inner DP loop, turning order-dependent computations from a one-time preprocessing step into repeated per-state operations.
Precomputing order-dependent rates thus represents an efficient compilation of the enabling semantics into a total order. This approach preserves correctness while enabling a substantially faster evaluation of each candidate ordering.

6.5.2 Branch & Bound

This decomposition enables the use of a *branch-and-bound* (B&B) strategy to obtain globally optimal solutions.
The algorithm starts from the initial topological order derived from the observed timestamps.
Once an order is fixed, the DP procedure computes the optimal aligned timestamps and the corresponding objective value, which provides an initial global upper bound. The search then proceeds recursively over concurrency groups; at each step, all permutations of the transitions within the current group are enumerated, subject to precedence constraints. For each feasible order, the exact cost is computed via DP. If the resulting cost improves the incumbent solution, it is updated. By exhaustively exploring all feasible permutations, the method guarantees convergence to the global optimum.

Complexity Analysis

In the worst case, the complexity is exponential in the size of the concurrency groups, due to the enumeration of all permutations of concurrently enabled transitions. Let n denote the total number of transitions. For multiple concurrency groups of sizes $c_1, c_2, \ldots, c_g$, the total number of candidate topological orders is $M = c_1! \cdot c_2! \cdots c_g!$, and the overall runtime is of the order $O(M \cdot nk)$, where k is the number of candidate timestamps per transition used in the DP computation. Here, n contributes linearly per DP evaluation, since the DP must optimize timestamps for all n events in each candidate order, while the factorial growth of each c_i dominates the combinatorial explosion in the number of orders.
Precomputing order-dependent transition rates for each candidate topological order requires only a single linear pass over the order and has negligible cost compared to the DP evaluation. For small sized concurrency groups, this approach is tractable and guarantees the global optimum.

Algorithm 4 Global B&B for partial-order alignment

Require: Observed timestamps $\hat{\theta}$, rates Γ, weights α, β, predecessor lists, concurrent groups
Ensure: Optimal timestamps $\theta^\star$, minimal cost, best event order
auxiliary functions Initialize topological order, compute initial DP cost, set global best

```
1: procedure BRANCH(curr_order, groups_remaining)
2:     if groups_remaining empty then
3:         Evaluate DP alignment for curr_order
4:         if cost < global best then Update global best
5:             return
6:         end if
7:         Select next concurrent group G_i
8:         for each feasible permutation of G_i do
9:             Replace G_i in curr_order with permutation
10:            BRANCH(new order, remaining groups)
11:        end for
12:
13:        Run BRANCH(initial order, all groups)
14:        return global best θ*, cost, order
```

As the degree of concurrency grows, however, exhaustive enumeration quickly becomes infeasible.

6.5.3 A* Search

As an alternative, the exploration of admissible topological orders can be formulated as a shortest-path problem and solved using an *A* search* strategy. A related idea was proposed by Li, Polyvyanyy, and Leemans [11], although their setting differs from ours, as temporal aspects are not taken into account. In this approach, each search state corresponds to a partial topological order

$$\boldsymbol{\pi}^{(k)} = (\pi_1, \ldots, \pi_k),$$

which satisfies all precedence constraints within the prefix. For each state, we define:

- $g(\boldsymbol{\pi}^{(k)})$: the exact optimal cost of aligning the prefix;
- $h(\boldsymbol{\pi}^{(k)})$: a lower bound on the cost of aligning the remaining events;
- $f(\boldsymbol{\pi}^{(k)}) = g(\boldsymbol{\pi}^{(k)}) + h(\boldsymbol{\pi}^{(k)})$.

The heuristic function h must be *admissible*, which means it must never overestimate the true remaining cost.

Prefix Cost The prefix cost is defined as

$$g(\boldsymbol{\pi}^{(k)}) = \min_{\theta_1,\ldots,\theta_k} \alpha \sum_{i=1}^{k} W_{\pi_i}(\theta_i - \theta_{i-1}) + \beta \sum_{i=1}^{k} |\theta_i - \hat{\theta}_i|, \tag{6.9}$$

subject to $\theta_i \geq \theta_{i-1}$. This cost is computed exactly using the same DP procedure as for the full fixed-order problem 4.3.2, restricted to the prefix.

Admissible Heuristic For a partial prefix $\boldsymbol{\pi}^{(k)}$, let

- $S(\boldsymbol{\pi}^{(k)})$ denote the set of scheduled transitions, and
- $U(\boldsymbol{\pi}^{(k)})$ denote the set of unscheduled transitions.

For each remaining transition $i \in U(\boldsymbol{\pi}^{(k)})$, we define the earliest enabling time based on observed timestamps as

$$e_i(\boldsymbol{\pi}^{(k)}) = \max_{j \in \mathrm{pred}(i) \cap S(\boldsymbol{\pi}^{(k)})} \hat{\theta}_j, \tag{6.10}$$

with $e_i(\boldsymbol{\pi}^{(k)}) = 0$ if the intersection is empty.
The heuristic contribution of transition i is then

$$h_i(\boldsymbol{\pi}^{(k)}) = \beta \max\Big(0, e_i(\boldsymbol{\pi}^{(k)}) - \hat{\theta}_i\Big). \tag{6.11}$$

The total heuristic estimate is obtained by summation:

$$h(\boldsymbol{\pi}^{(k)}) = \sum_{i \in U(\boldsymbol{\pi}^{(k)})} h_i(\boldsymbol{\pi}^{(k)}). \tag{6.12}$$

This heuristic is admissible because it ignores likelihood terms and interactions between unscheduled transitions, and therefore cannot overestimate the true remaining cost.

Complexity Analysis

Let n denote the number of transitions and k the number of candidate timestamps. From Section 4.4, a single DP solve has complexity $O(nk)$. If E denotes the number of expanded states, the total complexity of A* is $O(Enk)$.
To improve scalability, we also consider *beam search*, which limits the number of active states to a fixed beam width B. Since the maximum search depth is n, the number of expanded states is bounded by $E = O(Bn)$, yielding an overall complexity of $O(Bn^2k)$.
While beam search sacrifices global optimality guarantees, it provides an effective trade-off between solution quality and runtime in large-scale settings.

Algorithm 5 Global A* / Beam Search for Partial-Order Alignment

Require: Observed timestamps $\hat{\theta}(1{:}n)$, rates $\Gamma(1{:}n)$, weights α, β, predecessor lists $\mathrm{Pred}(1{:}n)$, search mode `mode` $\in$ {'astar', 'beam'}, beam width B, maximum expansions $N_{\max}$

Ensure: Optimal aligned timestamps $\theta^\star(1{:}n)$, minimal cost $C^\star$, best topological order $O^\star$

Initialize root node: empty order O, placed events P, $g = 0$, $h =$ heuristic estimate, $f = g + h$
Open set, Closed set
Compute initial incumbent solution using topological order and DP alignment; set $\theta^\star, C^\star, O^\star$
while Open set is not empty **and** expansions $< N_{\max}$ **do**
    Select node with minimal f from Open set; remove it
    if node represents complete topological order **then**
        Evaluate DP alignment for full order
        if cost $< C^\star$ **then**
            Update incumbent solution: $\theta^\star, C^\star, O^\star$
        end if
        if mode = 'astar' **then break**
            continue
        end if
        Generate feasible successor events respecting predecessors
        for each successor event e **do**
            Extend current order: $O_{\text{new}} = [O, e]$
            Compute partial cost g_{new} via DP alignment on placed events
            Compute heuristic h_{new} for remaining events
            $f_{\text{new}} = g_{\text{new}} + h_{\text{new}}$
            if $f_{\text{new}} < C^\star$ **then**
                Add new node to Open set
            end if
        end for
        if mode = 'beam' **and** size(Open set) $> B$ **then**
            Keep only top B nodes with smallest f in Open set
        end if
        Mark current node as expanded in Closed set

        return $\theta^\star, C^\star, O^\star$

6.5.4 Topological Order Heuristic

As discussed in Chapter 5, when the process model does not perfectly fit the observed trace, some transitions may lack timestamps.
Both the Branch&Bound and A* alignment algorithms therefore begin by computing a *topological order* of transitions that is consistent with the model's precedence constraints and the available temporal information. Since some transitions may be silent, a specific strategy is required to order concurrently enabled transitions.
We compute this order using a timestamp-aware variant of Kahn's algorithm [45]. At each step, the algorithm considers the set of *enabled* transitions, i.e., transitions whose predecessors have already been assigned:

$$E_m = \{\, t_i \mid \text{all predecessors of } t_i \text{ have been assigned at step } m \,\}.$$

Among concurrently enabled transitions, we greedily prioritize those with earlier observed enabling times according to the score

$$\text{score}(t_i) = \begin{cases} \hat{\theta}_{t_i}, & \text{if an observed timestamp is available,} \\ -\infty, & \text{if } \hat{\theta}_{t_i} = \text{NaN (silent).} \end{cases} \tag{6.13}$$

The transition selected at step m is then

$$t^* = \arg\min_{t_i \in E_m} \text{score}(t_i),$$

which follows the standard greedy selection rule for topological sorting [46]. Silent transitions are thus scheduled as early as possible, while observed transitions are ordered consistently with their timestamps.

6.5.5 A* vs Branch & Bound

Both A* search and Branch & Bound address the combinatorial explosion caused by concurrency, but they are suitable for different problem scales.
For small and limited concurrency groups, Branch & Bound is particularly effective and guarantees global optimality. Moreover, B&B can serve as a reliable reference solution in experiments, providing exact results against which approximate methods can be compared. As the degree of concurrency increases, however, the number of feasible topological orders grows factorially, making exhaustive enumeration computationally infeasible. In such cases, A* offers a more scalable alternative by incrementally exploring the space of partial topological orders and pruning suboptimal prefixes using admissible heuristic bounds. Unlike B&B, which evaluates complete orders, A* leverages lower bounds on partial sequences to guide the search toward promising regions of the solution space. This makes A* particularly suitable for larger instances, where early pruning is essential and exact global search is impractical.

Part III

Experiments

Chapter 7

Experiments

All experiments were conducted on an Apple Mac equipped with an Apple M4 CPU (10 cores) and 16 GB of RAM. All algorithms were implemented in *MATLAB_R2025a*. In particular, the dynamic programming method introduced in Chapter 4, as well as the A* and branch-and-bound algorithms introduced in Chapter 6, were implemented in MATLAB. Linear programs were solved using MATLAB's *linprog*, while non-linear optimization problems were handled using *fmincon*.
Further implementation details are provided in Appendix A.

7.1 Fixed-Order Alignments

We computed the optimal alignments for the example in Chapter 4, Section 4.3 using a grid of 10,000 values of α, to precisely identify the parameter values of α and β at which the optimal solution changes.
The LP solver converged in approximately 4.66 s using the dual-simplex method, while the DP approach required 0.21 s with a naïve implementation, which was further reduced to 0.11 s by exploiting prefix minima.
We then applied the same procedure to the hospital example from Chapter 5, Section 5.1. Using the same α grid, the LP solver required 5.93 s, while DP took 0.27 s in the naïve version and 0.10 s with prefix minima. In all cases, LP and DP produced identical aligned timestamps, reported in the Table 7.1.
These results are fully consistent with Proposition 1, as each optimal solution is composed of timestamps drawn from the finite set of observed timestamps, and changes in the optimal alignment occur only at the parameter values where two extreme points exchange optimality.
As in the example in Section 4.3, for $\alpha = 0$ the optimal timestamps coincide with the observed ones. Particular attention should be paid to the timestamp of the

Table 7.1: Optimal alignments for the hospital example.

α	β	Optimal Alignment
0.000,0	1.000,0	(τ_2,0), (BT,1.123), (PT,1.228), (SC,4.682), (S,73.72), (FC,164.8)
0.978,9	0.021,1	(τ_2,0), (BT,0), (PT,1.228), (SC,4.682), (S,73.72), (FC,164.8)
0.991,0	0.009,0	(τ_2,0), (BT,0), (PT,1.228), (SC,4.682), (S,164.8), (FC,164.8)
0.996,2	0.003,8	(τ_2,0), (BT,0), (PT,1.228), (SC,1.228), (S,164.8), (FC,164.8)
0.996,8	0.003,2	(τ_2,0), (BT,0), (PT,0), (SC,0), (S,164.8), (FC,164.8)

silent transition, which has no observed reference; its optimal timestamp is 0. This outcome is intuitive: the silent transition has a high rate, making it nearly immediate. Moreover, since silent transitions are not penalized by the timestamp deviation term, the objective reduces to maximizing the likelihood, which is achieved by minimizing the difference between the timestamp of the silent transition and that of the preceding event (here, time 0 of the first consult).
The first change in the optimal alignment occurs at $\alpha = 0.9789$, indicating that the timestamp deviation term dominates the objective over a wide range of parameters, while the likelihood term affects the solution only for values of α close to 1. This behaviour can be attributed to the long waiting times in the example, which amplify the contribution of the distance term. The choice of hours as the time unit does not significantly affect the relative weight of this component, as we obtained identical results when using minutes.
When $\alpha = 1$, the objective reduces to pure likelihood maximization. In this case, the entire waiting time is allocated to the transition with the smallest rate, namely *Surgery*.
These experiments confirm that the DP algorithm yields the same optimal alignments as LP with significantly lower runtime.

7.2 Partial-Order Alignments

We next evaluated the A* algorithm on the motivating example shown in Figure 4.1, considering the log trace

$$\langle (a,12.3), (c,12.5), (b,13.2), (d,19.1) \rangle.$$

To illustrate the algorithm's behaviour, we report the evolution of the search tree and pruning decisions for $\alpha = 0.918919$.
Since the number of concurrently unordered events is small, we can directly compare A* with the Branch & Bound approach. In this example, a single concurrent group consists of two transitions (b and c), resulting in only $2! = 2$ admissible permutations.

```
[EXPAND] depth=0 | order=[] | g=0.0000 h=0.0000 f=0.0000
  -> try add event 1 -> 1

[EXPAND] depth=1 | order=1 | g=1.1303 h=0.0000 f=1.1303
  -> try add event 2 -> [1 2]
  -> try add event 3 -> [1 3]

[EXPAND] depth=2 | order=[1 3] | g=0.9973 h=0.0000 f=0.9973
  -> try add event 2 -> [1 3 2]

[EXPAND] depth=2 | order=[1 2] | g=1.2859 h=0.0000 f=1.2859
  -> try add event 3 -> [1 2 3]

[EXPAND] depth=3 | order=[1 2 3] | g=1.3427 h=0.0000 f=1.3427
  -> try add event 4 -> [1 2 3 4]

[EXPAND] depth=3 | order=[1 3 2] | g=1.3427 h=0.0000 f=1.3427
  -> try add event 4 -> [1 3 2 4]
     pruned: g >= best (3.3200 >= 3.3200)

[EXPAND] depth=4 | order=[1 2 3 4] | g=2.9108 h=0.0000 f=2.9108

[COMPLETE ORDER FOUND] [1 2 3 4]
  full cost  = 2.910827
  theta      = [13.2 13.2 19.1 19.1]

[NEW BEST ALIGNMENT]
  order = [1 2 3 4]
  cost  = 2.910827
  theta = [13.2 13.2 19.1 19.1]
```

Figure 7.1: Evolution of the A* search tree for the example shown in Figure 4.1 with $\alpha = 0.918919$.

Thus, exhaustive enumeration is feasible, and B&B is guaranteed to return the global optimum.
The mean runtime of A* was 2.41 s, compared with 2.17 s for B&B, using a grid of 1,000 values of α. Here, B&B is slightly faster due to the small problem size. For larger instances with more events and larger concurrency groups, A* is expected to scale better due to its pruning capabilities.
Table 7.2 reports the optimal alignments for selected values of α.

Table 7.2: Optimal timestamps and event order for selected α values.

α	β	Optimal Alignment
0.000,0	1.000,0	(a,12.3000), (c,12.5000), (b,13.2000), (d,19.1000)
0.485,0	0.515,0	(a,12.5000), (c,12.5000), (b,13.2000), (d,19.1000)
0.596,0	0.404,0	(a,13.2000), (b,13.2000), (c,19.1000), (d,19.1000)
0.960,0	0.040,0	(a,19.1000), (c,19.1000), (b,19.1000), (d,19.1000)

We also solved the alignment problem using a nonlinear solver; in this case, the mean runtime was 1.77 s, using the same grid of 1,000 values of α. While the results are qualitatively similar to those obtained with the combinatorial methods (B&B and A*), small quantitative differences are observed.

Table 7.3: Optimal timestamps obtained with the nonlinear solver

α	β	Optimal Alignment
0.000,0	1.000,0	(a,12.3), (b,13.2), (c,12.5), (d,19.1)
0.738,7	0.261,3	(a,13.2), (b,13.2), (c,19.1), (d,19.1)
0.748,7	0.251,3	(a,13.27), (b,13.27), (c,13.27), (d,19.1)
0.749,7	0.250,3	(a,13.2), (b,13.2), (c,19.1), (d,19.1)
0.755,8	0.244,2	(a,13.29), (b,13.29), (c,13.29), (d,19.1)
0.756,8	0.243,2	(a,13.2), (b,13.2), (c,19.1), (d,19.1)
0.952,0	0.048,0	(a,19.1), (b,19.1), (c,19.1), (d,19.1)

A notable difference observed with the nonlinear solver is the appearance of sudden jumps in the aligned timestamps around $\alpha \approx 0.75$. In particular, the optimal timestamp of transition c oscillates between 13.27, 13.29, and 19.1. Recall that 13.2 is the optimal timestamp obtained in chapter 6 solving the Timed Stochastic Alignment Problem 4.3.2 considering the observed order "*acbd*", whereas 19.1 is obtained considering the order most likely order "*abcd*", which is also the solution returned by the heuristic algorithms. Numerically, oscillations between values 13.27 and 13.29 are negligible in terms of objective value and stem from the presence of a nearly flat region in the objective landscape: several timestamp configurations

yield almost identical costs. This indicates numerical sensitivity of the continuous optimization procedure in a nonsmooth and non-strictly convex setting. Small variations in α may cause the optimization procedure to switch from one plateau to another, yielding timestamps that do not exactly coincide with the observed values. Such behaviour is typical in nonsmooth optimization problems where the objective function involves maximum operators.

The discrepancies between the nonlinear solver and the combinatorial methods (B&B and A*) can be attributed to several factors. B&B and A* operate over a discretized set of candidate timestamps derived from the observations, thereby guaranteeing a global optimum on that finite grid. In contrast, the nonlinear solver explores the continuous space, allowing slight deviations from the observed timestamps. Furthermore, nonlinear optimization methods may converge to different local minima depending on initialization or tolerance settings, whereas B&B and A* identify the global optimum within the discretized domain. Continuous evaluation of the objective may also introduce minor numerical rounding effects, especially when multiple timestamp configurations yield nearly identical objective values. Additionally, for some values of α, even small changes can noticeably shift the trade-off between following the observed timestamps and maximizing the model likelihood, causing sudden changes in the solver's solutions.

Despite these differences, the overall behaviour across α remains consistent with the results reported in Table 7.2. As expected, for small α the alignment adheres closely to the observed timestamps, while for larger α the solution progressively shifts toward the model's most likely temporal configuration. The mean runtime of the nonlinear solver was 1.77 s, slightly faster than B&B and A* for this small instance, indicating that continuous optimization can be computationally competitive while capturing the same qualitative trends. Nevertheless, unlike the combinatorial approaches, it does not provide a formal guarantee of global optimality.

Finally, we applied both heuristic algorithms and the nonlinear solver to the larger example from Chapter 5, Section 5.1, using an α grid of 100 values. In this case, the optimal alignment coincides with the one obtained in the fixed-order setting. The mean runtime was 0.90 s for A*, 1.33 s for B&B and 1.23 s for the nonlinear solver. This confirms that A* scales more favorably as the number of events and concurrent groups increases.

In particular, the results obtained using the nonlinear solver are reported in Table 7.4.

The nonlinear solver produces a similar overall pattern, but small numerical artifacts are visible. For instance, for large α, some timestamps take values extremely close to zero (of the order of 10^{-20}), reflecting that the solver explores nearly flat regions of the objective function where multiple configurations have almost the same cost. These tiny deviations highlight the sensitivity of continuous optimization compared with the discrete DP or combinatorial methods, which restrict candidate

Table 7.4: Optimal alignments for the hospital example obtained with the nonlinear solver.

α	β	Optimal Alignment
0.000,0	1.000,0	(τ_2, 0), (BT, 1.123), (PT, 1.228), (SC, 4.682), (S, 73.72), (FC, 164.8)
0.986,0	0.014,0	(τ_2, 0), (BT, 3.337×10^{-21}), (PT, 1.228), (SC, 4.682), (S, 73.72), (FC, 164.8)
0.991,0	0.009,0	(τ_2, 0), (BT, 7.879×10^{-22}), (PT, 1.227), (SC, 4.682), (S, 164.8), (FC, 164.8)
0.992,0	0.008,0	(τ_2, 0), (BT, 2.208×10^{-16}), (PT, 1.226), (SC, 4.677), (S, 164.8), (FC, 164.8)
0.993,0	0.007,0	(τ_2, 0), (BT, 3.095×10^{-20}), (PT, 1.227), (SC, 4.682), (S, 164.8), (FC, 164.8)
0.997,0	0.003,0	(τ_2, 2.22×10^{-16}), (BT, 2.906×10^{-24}), (PT, 2.22×10^{-16}), (SC, 2.22×10^{-16}), (S, 164.8), (FC, 164.8)

timestamps to a finite grid.
The remaining timestamps are nearly identical to the observed ones, differing only by a few hundredths. These small discrepancies arise from the continuous nature of the optimization, in contrast to the discrete grid used by the combinatorial methods.
Overall, the results confirm that the heuristic approaches (A* and B&B) produce identical optimal timestamps and event orders for the selected α values, demonstrating their reliability on the discrete grid. The nonlinear solver, while yielding a very close approximation in this case, may exhibit small deviations due to local plateaus or numerical rounding. Importantly, unlike the combinatorial methods, it does not provide a guarantee of global optimality.

7.3 Reproducibility

All experiments were conducted using deterministic algorithms and fixed input data, ensuring that results are fully reproducible. All experiments were executed on a single machine with a fixed hardware configuration. No randomization was used in the optimization procedures or in the exploration of topological orders. All model parameters (transition rates, observed timestamps, and precedence relations) were explicitly specified, and the grids of α values were fixed for each experiment. For A* and beam search, the heuristic function and beam width were kept constant across runs. Given the same input traces, model parameters, and solver settings, the reported aligned timestamps and optimal orders can be reproduced exactly. Due to JIT compilation, memory management, and operating system scheduling,

runtimes exhibit small variations across runs. All reported runtimes are averaged over 10 executions and rounded to the nearest hundredth of a second.

Part IV

Conclusion

Chapter 8

Conclusion and Future Work

In this thesis we introduced a novel framework for aligning observed timed traces with timed stochastic process models. Our approach explicitly accounts for both the likelihood of a trace under the model and its temporal deviation from the observed timestamps, combining these two aspects through a tunable, parameterized objective function.

We began by considering the case of *perfectly fitting models*, where the model can exactly replay the observed timed trace. In this setting, we showed that the alignment problem can be formulated as a linear program, enabling the derivation of key structural properties. Notably, we proved that optimal timestamps can always be chosen from the set of observed timestamps, and that the optimal solution exhibits piecewise-constant behaviour with respect to the trade-off parameter between likelihood and temporal deviation.

We then extended the framework to *non-perfectly fitting models*, introducing a two-step procedure that combines time-aware alignment with classical untimed alignment techniques based on *model moves* and *log moves*.

Finally, we generalized the method to handle partial-order alignments and discussed heuristic strategies, such as Branch & Bound and A* based search, to efficiently address the combinatorial complexity introduced by partial orders.

The experimental evaluation provides strong empirical support for the theoretical framework and algorithms developed in this thesis.

For fixed-order alignments, dynamic programming consistently outperformed linear programming while returning identical optimal timestamps across all tested instances. This result confirms the theoretical characterization of optimal solutions as extreme points of the observed timestamp set and demonstrates the practical efficiency of the proposed DP formulation. For partial-order alignments, the combinatorial approaches (A* and B&B) reliably identified the global optimum over the discretized timestamp grid. While B&B performs well for small concurrency groups, A* demonstrated better scalability as the size and number of concurrent

groups increases.
These findings underline the suitability of informed search techniques for addressing partial-order constraints in timed stochastic conformance checking.
The nonlinear solver provides a computationally competitive continuous alternative and captures the same qualitative trade-off between timestamp deviation and model likelihood. However, unlike the combinatorial methods, it does not provide a formal guarantee of global optimality and may exhibit minor numerical artifacts or convergence to local minima due to the nonsmooth nature of the objective function.
To sum up, the experimental results successfully capture the intuitive notion of alignment between timed traces and stochastic process models, confirming that the proposed algorithms produce meaningful and reproducible solutions.

8.1 Future Work

Several promising directions for future research emerge from this work.
The first would be to unify the optimization framework for models that do not perfectly fit the observed data. The current two-step alignment procedure can be seen as a decomposition heuristic: it improves scalability but does not guarantee global optimality. A single, integrated approach that simultaneously accounts for structural, temporal, and stochastic deviations could overcome this limitation and provide stronger theoretical guarantees.
Scalability to larger datasets is another important direction. Applying these methods to bigger and more complex event logs will require access to richer stochastic Petri nets, which are not always easily available, as well as more efficient techniques for extracting temporal and stochastic information. Assessing performance and robustness on such datasets will be key to demonstrating the framework's practical applicability in real-world scenarios.
There is also an opportunity to extend the evaluation beyond the observed traces; there may be sequences of activities that are allowed by the model and consistent with its firing rates but are simply not observed in the data. Identifying these *likely but unobserved* traces, for example through stochastic simulation or model sampling, could provide a more complete picture of the model's *precision* and its ability to generalize.
Extending the alignment techniques to generalized stochastic Petri nets (GSPNs) would also be important. Relaxing the assumption of exponential delays and incorporating immediate transitions would increase the modeling expressiveness, enabling the framework to represent a wider variety of real-world processes.
Finally, the alignment framework developed in this thesis could serve as a foundation for stochastic timed process discovery. By enabling the automated extraction of probabilistic timed process models directly from data, this approach could open

new opportunities for analyzing complex processes in domains where timing and uncertainty play a critical role.

Overall, this thesis provide a solid foundation for advancing both the theory and practice of stochastic timed process analysis. By bridging probabilistic modeling and temporal reasoning, the proposed framework opens new opportunities to better understand, monitor, and discover complex real-world processes.

Appendix A

Matlab code

A.1 LP

This section reports the Matlab implementation of the linear programming used in Chapters 4, 7. The function leverages MATLAB's *linprog* function to solve the underlying linear program.

```
function Results = alignment(theta_obs, lambda_M, alpha_grid, A, b
    ↪ , Aeq, beq, lb, ub)
% ALIGNMENT Solve the timestamp alignment LP for a grid of alpha
    ↪ values using linprog.
%
% INPUTS
%   theta_obs     : observed timestamps (n x 1)
%   lambda_M   : rate parameters (n x 1)
%   alpha_grid: vector of alpha values in [0,1]
%   A, b       : inequality constraint matrices (A x <= b)
%   Aeq, beq   : equality constraint matrices (Aeq x = beq)
%   lb, ub     : lower and upper bounds for LP variables
%
% OUTPUT
%   Results    : struct array containing for each alpha:
%                  .alpha    - value of alpha
%                  .beta     - value of beta (=1-alpha)
%                  .t        - optimal timestamps
%                  .u        - slack variables
%                  .obj      - optimal LP value
%                  .method   - solver used
%                  .exitflag- solver exit flag
%


% basic checks
theta_obs    = theta_obs(:);
```

```
lambda_M = lambda_M(:);
n = numel(lambda_M );

beta_grid = 1 - alpha_grid;

% Solver setup
options = optimoptions('linprog', ...
    'Display','none', ...
    'Algorithm','dual-simplex', ...
    'MaxIterations',1e5);

% Preallocate result struct
Results = struct('alpha',cell(numel(alpha_grid),1), ...
                 'beta', [], 't', [], 'u', [], ...
                 'obj', [], 'method', [], 'exitflag', []);

% Precompute lambda_{i+1}
lambda_next = [lambda_M(2:end); 0];

% MAIN LOOP
for idx = 1:numel(alpha_grid)

    alpha = alpha_grid(idx);
    beta  = beta_grid(idx);

    % Build cost vector
    coeff_t = alpha * (lambda_M - lambda_next);   % size n
    coeff_u = beta  * ones(n,1);                  % size n
    f_vec   = [coeff_t; coeff_u];                 % size 2n

    % Solve LP
    [x, fval, exitflag] = linprog(f_vec, A, b, Aeq, beq, lb, ub,
    ↪ options);

    % Extract solution parts
    theta_star = x(1:n);
    u_star = x(n+1:end);

    % Store output
    Results(idx).alpha    = alpha;
    Results(idx).beta     = beta;
    Results(idx).t        = theta_star;
    Results(idx).u        = u_star;
    Results(idx).obj      = fval;
    Results(idx).method   = 'linprog-dual-simplex';
    Results(idx).exitflag = exitflag;
end
end
```

A.2 DP

This section reports the Matlab implementations of the dynamic programming algorithms used in Chapter 4,5, 6 and for the experiments of Chapter 7.

A.2.1 DP naïve

```
function [theta_opt, cost_opt] = function [theta_opt, cost_opt] =
    ↪ dp_alignment(timestamps, rate_M, alpha, beta)
% DP_ALIGNMENT
%
% INPUT:
%   timestamps    - observed timestamps (k x 1)
%   rate_M - rate parameters (k x 1)
%   alpha     - weight for likelihood
%   beta      - weight for timestamp distance
%
% OUTPUT:
%   theta_opt     - optimal aligned timestamps (k x 1)
%   cost_opt - minimum total cost


%setup
T = max(timestamps);

k = numel(rate_M);      % Number of events

%candidate grid of timestamps:
D = unique([0; timestamps(:)]);
D = sort(D(:))';       % Row vector
nD = numel(D);

% Dynamic programming cost table:
%   V(i,j) = minimal cost up to event i if theta_i = D(j)
V   = inf(k, nD);

% Pointer table for backtracking:
%   Ptr(i,j) = index of best predecessor at stage i-1
Ptr = zeros(k, nD);

%  INITIALIZATION (stage i = 1)

```

```
35
36 for j = 1:nD
37     t = D(j);
38
39     % likelihood for theta_0 = 0 (first step)
40     trans_cost = alpha * rate_M(1) * (t - 0);
41
42     % timestamp distance |t - timestamps(1)|
43     if isnan(timestamps(1))
44             obs_cost = 0;
45     else
46             obs_cost = beta * abs(t - timestamps(1));
47     end
48     % Total cost for stage 1
49     V(1,j) = trans_cost + obs_cost;
50
51     % No predecessor at i=1
52     Ptr(1,j) = 0;
53 end
54
55
56 %  DP RECURSION (stages i = 2,...,k)
57
58
59 for i = 2:k
60     for j = 1:nD
61         t = D(j);
62
63         % If this is the last stage, enforce theta_k >= T
64         if (i == k) && (t < T)
65             continue;   % skip infeasible terminal states
66         end
67
68         % Previous timestamps must satisfy monotonicity: D(prev)
    ↪ <= t
69         idx_prev = find(D <= t);
70
71         if isempty(idx_prev)
72             continue;   % no feasible predecessor
73         end
74
75         % Retrieve DP cost from previous stage
76         prev_costs = V(i-1, idx_prev);
77
78         %likelihood cost
79         transition_costs = alpha * rate_M(i) * (t - D(idx_prev));
80
81         % Observation cost
82         if isnan(timestamps(i))
```

```
            obs_costs = 0;
        else
            obs_costs = beta * abs(t - timestamps(i));
        end
        % Total cost
        costs = prev_costs + transition_costs + obs_costs;

        % Select best predecessor
        [minVal, relIdx] = min(costs);

        V(i,j)   = minVal;
        Ptr(i,j) = idx_prev(relIdx);
    end
end

%  FINAL SELECTION

% Last timestamp must satisfy theta_k >= T
idx_final = find(D >= T);

[cost_opt, rel] = min(V(k, idx_final));
idx_min = idx_final(rel);


%  BACKTRACKING to recover optimal sequence of timestamps

theta_opt = zeros(k,1);
theta_opt(k) = D(idx_min);

idx = idx_min;
for i = k:-1:2
    idx = Ptr(i, idx);    % move one stage back
    theta_opt(i-1) = D(idx);
end

end
```

A.2.2 DP with prefix minima

```
function [theta_opt, cost_opt] = dp_alignment_fast(timestamps,
    ↪ rate_M, alpha, beta)
% DP_ALIGNMENT_FAST
% INPUT:
%   timestamps   : observed timestamps (k x 1), NaN = silent
    ↪ transition
%   rate_M  : transition rates (k x 1)
```

```
%   alpha    : weight for model cost (likelihood)
%   beta     : weight for observation deviation
%
% OUTPUT:
%   theta_opt    : optimal aligned timestamps (k x 1), optimal even
    ↪  for NaNs
%   cost_opt: minimal total cost

% Input validation & setup
k = numel(rate_M);
timestamps = timestamps(:);           % force column vector
assert(numel(timestamps) == k, 'timestamps and rate_M must have
    ↪ same length');


obs_times = timestamps(~isnan(timestamps));
D_raw = unique([0; obs_times]);
D = sort(D_raw(:));          % sorted unique timestamps
n = numel(D);

T_end = D(end);


% DP tables
V_prev = inf(1, n);
V_curr = inf(1, n);
Ptr = zeros(k, n);           % backpointers

% Stage 1: Initialize first transition
for j = 1:n
    t = D(j);
    transCost = alpha * rate_M(1) * t;

    if isnan(timestamps(1))
        obsCost = 0;
    else
        obsCost = beta * abs(t - timestamps(1));
    end

    V_prev(j) = transCost + obsCost;
    Ptr(1, j) = 0;               % no predecessor for stage 1
end

% DP Recursion: stages i=2 to k
for i = 2:k
    V_curr(:) = inf;             % reset current stage

    % Prefix minima optimization (convex cost structure)
    A = V_prev - alpha * rate_M(i) * D';    % precompute for all j
```

```
    M = zeros(1, n); I = zeros(1, n);        % minima and argmin trackers
    M(1) = A(1); I(1) = 1;
    for r = 2:n
        if A(r) < M(r-1)
            M(r) = A(r); I(r) = r;
        else
            M(r) = M(r-1); I(r) = I(r-1);
        end
    end

    % Fill current stage
    for j = 1:n
        t = D(j);

        % Terminal constraint
        if i == k && ~isempty(T_end) && t < T_end
            continue;
        end

        % Transition + observation costs
        transCost = M(j) + alpha * rate_M(i) * t;

        if isnan(timestamps(i))
            obsCost = 0;
        else
            obsCost = beta * abs(t - timestamps(i));
        end

        V_curr(j) = transCost + obsCost;
        Ptr(i, j) = I(j);            % backpointer to previous stage
    end

    % Check if stage is feasible
    if all(isinf(V_curr))
        error('Stage i=%d infeasible: all V_curr = Inf', i);
    end

    V_prev = V_curr;
end

% Termination: find best final state
valid_idx = find(D >= T_end);
if isempty(valid_idx)
    error('No valid terminal states: all D < T_end=%.6f', T_end);
end

if all(isinf(V_prev(valid_idx)))
```

```
    error('All terminal states Inf: no feasible path');
end

[cost_opt, rel_idx] = min(V_prev(valid_idx));
j_star = valid_idx(rel_idx);

% Backtracking: recover optimal path
theta_opt = zeros(k, 1);
j = j_star;
for i = k:-1:1
    theta_opt(i) = D(j);
    if i > 1
        j = Ptr(i, j);
    end
end

end
```

A.3 Nonlinear-solver

This section reports the Matlab implementation of the non-linear program presented in Chapter 6, and used in the experiments of Chapter 7.

```


function [theta_star, e_star, fval, exitflag, output] =
    ↪ optimize_po_trace( ...
        Gamma, theta_hat, pred_lists, alpha, beta)

    n = numel(Gamma);

    % Initial guess
    theta0 = theta_hat(:);
    e0 = zeros(n,1);

    for i = 1:n
        idx = pred_lists{i};
        if ~isempty(idx) && isnumeric(idx)
            e0(i) = max(theta_hat(idx));
        end
    end
    x0 = [theta0; e0];

    % Variable bounds
    lb = zeros(2*n,1);
    ub = [];
```

```
23
24      % Linear constraint: theta_n >= theta_hat(end)
25      A = zeros(1, 2*n);
26      A(1,n) = -1;          % -theta_n <= -theta_hat(end)
27      b = -theta_hat(end);
28      Aeq = []; beq = [];
29
30      % Precompute predecessor incidence matrix
31      PredMat = false(n);  % n x n logical matrix
32      for i = 1:n
33          preds = pred_lists{i};
34          if ~isempty(preds)
35              PredMat(i, preds) = true;
36          end
37      end
38
39      % Objective function
40      fun = @(x) objective_po(n, x, Gamma, theta_hat, alpha, beta);
41
42      % Nonlinear constraints (vectorized)
43      nonlcon = @(x) nonlcon_po(n, x, PredMat);
44
45      % Optimization options
46      options = optimoptions('fmincon', 'Algorithm', 'sqp', 'Display
    ↪ ', 'off');
47
48      % Run fmincon
49      [x_opt, fval, exitflag, output] = fmincon(fun, x0, ...
50                                                A, b, Aeq, beq, lb,
    ↪ ub, ...
51                                                nonlcon, options);
52
53      % Unpack solution
54      theta_star = x_opt(1:n);
55      e_star     = x_opt(n+1:end);
56
57  end
```

A.3.1 Auxiliary functions

This subsection reports the auxiliary function that computes the objective function.

```
1   function f = objective_po(n,x, Gamma, theta_hat, alpha, beta)
2
3       theta = x(1:n);
4       e     = x(n+1:end);
5
```

```
6       % likelihood term: alpha * sum  _i  ( _i  - e_i)
7       term1 = alpha * sum(Gamma(:) .* (theta(:) - e(:)));
8
9       % timestamp deviation term: beta * sum | _i  - \hat  _i |
10      term2 = beta * sum(abs(theta(:) - theta_hat(:)));
11
12      f = term1 + term2;
13  end
14  function [c, ceq] = nonlcon_po(n, x, PredMat)
15
16      theta = x(1:n);
17      e     = x(n+1:end);
18
19      % Inequalities: e_i <= theta_i
20      c = e - theta;
21
22      % Equalities: e_i = max_j(theta_j), j in Pred(i)
23      ceq = zeros(n,1);
24
25      % Identify rows with predecessors
26      hasPreds = any(PredMat, 2);
27
28      % For rows with predecessors: compute max
29      if any(hasPreds)
30          theta_mat = repmat(theta(:)', n, 1); % n x n
31          theta_mat(~PredMat) = -Inf;          % ignore non-
    ↪ predecessors
32          ceq(hasPreds) = e(hasPreds) - max(theta_mat(hasPreds,:),
    ↪ [], 2);
33      end
34
35      % Rows without predecessors: e_i = 0
36      ceq(~hasPreds) = e(~hasPreds);
37  end
```

A.4 B&B

This section presents the MATLAB implementation of the global Branch-and-Bound (B&B) algorithm presented in Chapter 6, and used in the experiments of Chapter 7 to compute optimal partial-order alignments. The algorithm explores alternative linearizations of concurrent transitions and evaluates each candidate order using a dynamic programming (DP) alignment procedure. A global upper bound is maintained to prune suboptimal branches.

```
function [theta_best, cost_best, order_best] = dp_bb_global(
    ↪ timestamps, Gamma, alpha, beta, predecessors, groups)
% DP_BB_GLOBAL - Global branch-and-bound over all concurrent
    ↪ groups
%
% INPUT:
%   timestamps        : observed timestamps (k x 1)
%   Gamma             : rate parameters (k x 1)
%   alpha, beta       : weights for likelihood and timestamp distance
%   predecessors      : cell array of predecessors for each event
%   groups            : cell array of concurrent groups (indices of
    ↪ events)
%
% OUTPUT:
%   theta_best        : aligned timestamps minimizing the DP objective
%   cost_best         : minimum cost
%   order_best        : event order achieving the minimum

k = numel(timestamps);

% Initial topological order
order_init = topological_order(timestamps, predecessors);

% Compute rates for initial order
rate_M_init = calculate_rate(order_init, Gamma, predecessors);
% Compute DP for initial order
[t_init, cost_init] = dp_alignment_fixed_order_fast(timestamps,
    ↪ rate_M_init, alpha, beta, order_init);

theta_init = zeros(k,1);
theta_init(order_init) = t_init;

% Initialize global best
global_best.cost  = cost_init;
global_best.theta = theta_init;
global_best.order = order_init;

% Start recursive BB search
global_best = branch_bb_global(order_init, groups, timestamps,
    ↪ Gamma, alpha, beta, predecessors, global_best);

% Return best solution
theta_best = global_best.theta;
cost_best  = global_best.cost;
order_best = global_best.order;
end

% Recursive BB
```

```
function global_best = branch_bb_global(curr_order,
    ↪ groups_remaining, timestamps, Gamma, alpha, beta,
    ↪ predecessors, global_best)

% Base case: all groups processed
if isempty(groups_remaining)
    rate_M = calculate_rate(curr_order, Gamma, predecessors);
    [t_candidate, cost_candidate] = dp_alignment_fixed_order_fast(
    ↪ timestamps, rate_M, alpha, beta, curr_order);
    t_full = zeros(size(timestamps));
    t_full(curr_order) = t_candidate;

    if cost_candidate < global_best.cost
        global_best.cost  = cost_candidate;
        global_best.theta = t_full;
        global_best.order = curr_order;
    end
    return
end

% Take next concurrent group
Gi = groups_remaining{1};
remaining_groups = groups_remaining(2:end);
idx_in_order = find(ismember(curr_order, Gi));
n = numel(Gi);


candidate_orders = [];
candidate_costs  = [];
perms_Gi = perms(Gi);
% Evaluate each candidate permutation
for p = 1:size(perms_Gi,1)
    permuted_group = perms_Gi(p,:);
    candidate_order = curr_order;
    candidate_order(idx_in_order) = permuted_group;

    % Feasibility check wrt predecessors
    feasible = true;
    for i = 1:n
        e = permuted_group(i);
        preds = setdiff(predecessors{e}, Gi);
        idx_e = find(candidate_order==e);
        for pred = preds
            idx_pred = find(candidate_order==pred);
            if idx_pred > idx_e
                feasible = false;
                break;
            end
        end
```

```
90          if ~feasible, break; end
91      end
92
93      if feasible
94          rate_M = calculate_rate(candidate_order, Gamma,
    ↪ predecessors);
95          [~, cost_pi] = dp_alignment_fixed_order_fast(timestamps,
    ↪ rate_M, alpha, beta, candidate_order);
96          candidate_orders = [candidate_orders; candidate_order];
97          candidate_costs  = [candidate_costs; cost_pi];
98      end
99  end
100
101
102 % Recurse on remaining groups
103 for i = 1:size(candidate_orders,1)
104     global_best = branch_bb_global(candidate_orders(i,:),
    ↪ remaining_groups, timestamps, Gamma, alpha, beta,
    ↪ predecessors, global_best);
105 end
106 end
```

A.4.1 Auxiliary Functions

This subsection introduces the auxiliary procedures used by the global Branch-and-Bound algorithm.

The following function computes an initial topological order of events that is consistent with the causal constraints of the model. It is a timestamp-aware variant of Kahn's algorithm, where among all currently enabled events, the one with the smallest observed timestamp is selected. Events with missing timestamps (`NaN`) are treated as having priority, ensuring that silent transitions are scheduled as early as possible.

```
1  function order = topological_order(timestamps, predecessors)
2  % Time-aware version of Kahn's algorithm picking earliest
       ↪ available events
3  % Handles NaN timestamps by treating them as "earliest possible" (
       ↪ score = -Inf)
4  % Among concurrent candidates, greedily picks smallest observed
       ↪ time.
5  %
6  % INPUT:
7  %   timestamps        : observed timestamps (k x 1), NaN = missing
8  %   predecessors : cell array {1xk} where predecessors{i} = list
       ↪ of pred events
```

```
9
10 k = numel(timestamps);
11 timestamps = timestamps(:);
12
13 order = zeros(1, k);
14 assigned = false(1, k);
15
16 for step = 1:k
17     candidates = [];
18
19     % Find all enabled events
20     for i = 1:k
21         if assigned(i), continue; end
22
23         preds = predecessors{i};
24         if isempty(preds) || all(assigned(preds))
25             candidates(end+1) = i;
26         end
27     end
28
29
30     % Priority: pick candidate with smallest effective timestamp
31     scores = timestamps(candidates);
32     scores(isnan(scores)) = -Inf;           % NaN events first
33
34     [~, idx_best] = min(scores);            % smallest score wins
35     best_event = candidates(idx_best);
36
37     order(step) = best_event;
38     assigned(best_event) = true;
39 end
40 end
```

The next function implements the core recursive Branch-and-Bound logic. At each recursion level, one concurrent group is selected and all feasible permutations of its events are generated. Each permutation is checked for causal feasibility with respect to predecessor constraints and evaluated using the DP alignment procedure. If all concurrent groups have been processed, the function computes the final alignment cost for the resulting total order and updates the global best solution if an improvement is found.

```
1 function global_best = branch_bb_global(curr_order,
    ↪ groups_remaining, timestamps, Gamma, alpha, beta,
    ↪ predecessors, global_best)
2 if isempty(groups_remaining)
3     rate_M = calculate_rate(curr_order, Gamma, predecessors);
```

```
    [t_candidate, cost_candidate] = dp_alignment_fixed_order_fast(
    ↪ timestamps, rate_M, alpha, beta, curr_order);
    t_full = zeros(size(timestamps));
    t_full(curr_order) = t_candidate;
    if cost_candidate < global_best.cost
        global_best.cost  = cost_candidate;
        global_best.theta = t_full;
        global_best.order = curr_order;
    end
    return
end

Gi = groups_remaining{1};
remaining_groups = groups_remaining(2:end);
idx_in_order = find(ismember(curr_order, Gi));
n = numel(Gi);


perms_Gi = perms(Gi);


candidate_orders = [];
candidate_costs  = [];

for p = 1:size(perms_Gi,1)
    permuted_group = perms_Gi(p,:);
    candidate_order = curr_order;
    candidate_order(idx_in_order) = permuted_group;

    % Check feasibility wrt predecessors
    feasible = true;
    for i = 1:n
        e = permuted_group(i);
        preds = setdiff(predecessors{e}, Gi);
        idx_e = find(candidate_order==e);
        for pred = preds
            idx_pred = find(candidate_order==pred);
            if idx_pred > idx_e
                feasible = false;
                break;
            end
        end
        if ~feasible, break; end
    end
    if feasible
        rate_M = calculate_rate(candidate_order, Gamma,
    ↪ predecessors);
        [~, cost_pi] = dp_alignment_fixed_order_fast(timestamps,
    ↪ rate_M, alpha, beta, candidate_order);
```

```
        candidate_orders = [candidate_orders; candidate_order];
        candidate_costs  = [candidate_costs; cost_pi];
    end
end

% Recurse
for i = 1:size(candidate_orders,1)
    global_best = branch_bb_global(candidate_orders(i,:),
    ↪ remaining_groups, timestamps, lambda_M, alpha, beta,
    ↪ predecessors, global_best, beam_width);
end
end
```

Finally, the following function computes the race-aware firing rates for a given topological order. For each transition in the order, it determines the set of enabled transitions according to the causal structure and sums their rates.

```
function rate_M = calculate_rate(order, Gamma, predecessors)
% calculate_rate - compute race-aware total rates for a
   ↪ topological order
%
% INPUT:
%   order        : topological order of events (1 x k)
%   Gamma        : rates of all transitions (k x 1)
%   predecessors : cell array of predecessor sets for each
   ↪ transition
%
% OUTPUT:
%   rate_M        : race-aware rate at each event in the order

k = numel(order);
rate_M = zeros(k,1);

for i = 1:k
    t_i = order(i);

    % transitions not yet fired
    en_set = setdiff(1:k, order(1:i-1));

    % only transitions whose predecessors are already fired
    en_set = en_set(arrayfun(@(j) all(ismember(predecessors{j},
   ↪ order(1:i-1))), en_set));

    % sum of rates for all enabled transitions
    rate_M(t_i) = sum(Gamma(en_set));
end
end
```

A.5 A*

This section presents the MATLAB implementation of a global A* search algorithm presented in Chapter 6, and used in the experiments of Chapter 7 to compute optimal partial-order alignments. The algorithm explores the space of feasible topological orders incrementally, guided by a cost function composed of an exact partial alignment cost and an admissible heuristic estimating the remaining cost.

The same framework can be used as a beam search by limiting the size of the open set, trading optimality guarantees for computational efficiency.

```
function [theta_best, cost_best, order_best] = dp_astar_global( ...
    timestamps, Gamma, alpha, beta, predecessors,mode, beam_width, max_nodes,verbose)
% DP_ASTAR_GLOBAL - A* / Beam search over topological orders for global alignment
%
% INPUT:
%   timestamps        : observed timestamps (k x 1)
%   Gamma             : rate parameters (k x 1)
%   alpha, beta       : weights for likelihood and timestamp distance
%   predecessors      : cell array of predecessors for each event
%   mode              :'astar' or 'beam'
%   beam_width        : default 10
%   max_nodes         : default 10000
%   verbose           : true or false, default false

%
% OUTPUT:
%   theta_best        : aligned timestamps
%   cost_best         : minimal cost
%   order_best        : corresponding event order

if nargin<7
    beam_width=10;
end
if nargin<8
    max_nodes=10000;
end
if nargin<9
    verbose='false';
end
k = numel(timestamps);
```

```
31
32 % Step 1: Initial order (seed incumbent)
33 order_init = topological_order(timestamps, predecessors);
34 rate_M = calculate_rate(order_init, Gamma, predecessors);
35 [t_init, cost_init] = dp_alignment_fixed_order_fast( ...
36     timestamps, rate_M, alpha, beta, order_init);
37
38 theta_init = zeros(k,1);
39 theta_init(order_init) = t_init;
40
41 best_cost  = cost_init;
42 best_theta = theta_init;
43 best_order = order_init;
44
45
46
47 if verbose
48     fprintf('Initial cost (topological order): %.6f\n', best_cost)
   ↪ ;
49     fprintf('Initial order: %s\n', mat2str(best_order'));
50     fprintf('Initial theta: %s\n', mat2str(best_theta',4));
51 end
52
53 % Step 2: A* / Beam search on partial orders
54 open_set   = struct('order', {}, 'placed', {}, 'g', {}, 'h', {}, '
   ↪ f', {}, 'parent', {});
55 closed_map = containers.Map();
56
57 % Root node
58 root.order  = [];
59 root.placed = [];
60 root.g      = 0;
61 root.h      = compute_h_remaining([], timestamps,beta,
   ↪ predecessors, k);
62 root.f      = root.g + root.h;
63 root.parent = [];
64
65 open_set = root;
66 expanded = 0;
67 fprintf('alpha: %.6f\n', alpha);
68 while ~isempty(open_set)
69
70     % Pop best f
71     [~, idx_min] = min([open_set.f]);
72     curr = open_set(idx_min);
73     open_set(idx_min) = [];
74     expanded = expanded + 1;
75
76     if verbose
```

```
77          fprintf('\n[EXPAND] depth=%d | order=%s | g=%.4f h=%.4f f
    ↪ =%.4f\n', ...
78              numel(curr.order), mat2str(curr.order'), curr.g, curr.
    ↪ h, curr.f);
79      end
80
81      % Closed list check
82      key = order_key(curr.order);
83      if isKey(closed_map, key) && curr.g >= closed_map(key)
84          if verbose
85              fprintf('  skipped (closed list)\n');
86          end
87          continue;
88      end
89      closed_map(key) = curr.g;
90
91      if expanded > max_nodes
92          fprintf('Hit max nodes (%d), stopping.\n', max_nodes);
93          break;
94      end
95
96      % Goal test
97      if numel(curr.order) == k
98          if verbose
99              fprintf('\n[COMPLETE ORDER FOUND] %s\n', mat2str(curr.
    ↪ order'));
100         end
101
102         rate_M = calculate_rate(curr.order, Gamma, predecessors);
103         [t_full, cost_full] = dp_alignment_fixed_order_fast( ...
104             timestamps, rate_M, alpha, beta, curr.order);
105
106         theta_full = zeros(k,1);
107         theta_full(curr.order) = t_full;
108
109         if verbose
110             fprintf('  full cost  = %.6f\n', cost_full);
111             fprintf('  theta      = %s\n', mat2str(theta_full',4))
    ↪ ;
112         end
113
114         if cost_full < best_cost - 1e-12
115             best_cost  = cost_full;
116             best_theta = theta_full;
117             best_order = curr.order;
118             if verbose
119                 fprintf('\n[NEW BEST ALIGNMENT]\n');
120                 fprintf('  order = %s\n', mat2str(best_order'));
121                 fprintf('  cost  = %.6f\n', best_cost);
```

```
122                 fprintf('  theta = %s\n', mat2str(best_theta',4));
123             end
124         end
125         if strcmp(mode, 'astar')
126             break;
127         else
128             continue;
129         end
130     end
131
132     % Expand successors
133     succs = get_successors(curr.order, predecessors, k);
134
135     for next = succs(:)'
136         next_order  = [curr.order; next];
137
138         if verbose
139             fprintf('  -> try add event %d -> %s\n', next, mat2str
    ↪ (next_order'));
140         end
141
142         % g-cost
143         g_next = compute_g_partial(next_order, timestamps, Gamma,
    ↪ alpha, beta, predecessors);
144
145         if g_next >= best_cost
146             if verbose
147                 fprintf('     pruned: g >= best (%.4f >= %.4f)\n',
    ↪  g_next, best_cost);
148             end
149             continue;
150         end
151
152         % h-cost
153         h_next = compute_h_remaining(next_order, timestamps, beta,
    ↪  predecessors, k);
154         f_next = g_next +  h_next;
155
156         if f_next >= best_cost
157             if verbose
158                 fprintf('     pruned: f >= best (%.4f >= %.4f)\n',
    ↪  f_next, best_cost);
159             end
160             continue;
161         end
162
163         key_next = order_key(next_order);
164         if isKey(closed_map, key_next) && g_next >= closed_map(
    ↪ key_next)
```

```
            if verbose
                fprintf('     pruned: dominated in closed list\n')
    ↪ ;
            end
            continue;
        end

        s.order  = next_order;
        s.placed = next_order;
        s.g      = g_next;
        s.h      = h_next;
        s.f      = f_next;
        s.parent = curr;

        open_set(end+1) = s;
    end

    % Beam pruning
    if strcmp(mode, 'beam') && numel(open_set) > beam_width
        [~, idx_sorted] = sort([open_set.f]);
        open_set = open_set(idx_sorted(1:beam_width));
        if verbose
            fprintf('  beam pruned to %d states\n', beam_width);
        end
    end
end

theta_best = best_theta;
cost_best  = best_cost;
order_best = best_order;

fprintf('\nSearch done. Best cost: %.6f | expansions: %d\n',
    ↪ best_cost, expanded);
end
```

A.5.1 Auxiliary Functions

This subsection describes the auxiliary functions used by the A* and beam search algorithm.

The following function converts a (partial) order into a unique string representation. This encoding is used as a key in the closed list to detect and prune dominated states that have already been explored with a lower or equal cost.

```

function key = order_key(order_vec)
    if isempty(order_vec)
```

```
        key = '[]';
    else
        key = sprintf('%d-', order_vec);
    end
end
```

This function computes the successor states of a given partial order. A successor is obtained by appending any transition whose predecessors have all been placed in the current order. This ensures that only causally feasible extensions are explored by the search algorithm.

```

function successors = get_successors(curr_order, predecessors, k)
    successors = [];
    for i = 1:k
        if ~ismember(i, curr_order)
            if all(ismember(predecessors{i}, curr_order))
                successors(end+1) = i;
            end
        end
    end
end
```

The following function computes the exact cost associated with a partial order. It evaluates the DP alignment cost restricted to the placed events, using the race-aware rates induced by the current order prefix.

```
function g = compute_g_partial(partial_order, timestamps, Gamma,
    ↪ ...
                                alpha, beta, predecessors)
    n = numel(partial_order);
    if n == 0
        g = 0;
        return;
    end
    rate_M = calculate_rate(partial_order, Gamma(partial_order),
    ↪ predecessors);
    sub_order = 1:n;
    [~, g] = dp_alignment_fixed_order_fast( ...
        timestamps(partial_order), rate_M, alpha, beta, sub_order)
    ↪ ;
end
```

The final auxiliary function computes a heuristic lower bound on the cost of the events that have not yet been placed in the partial order. For each remaining transition, it estimates the minimal timestamp deviation required to satisfy its predecessor constraints.

```
function h = compute_h_remaining(placed, timestamps, beta,
    ↪ predecessors, k)
    remaining = setdiff(1:k, placed);
    h = 0;
    for i = remaining
        preds = predecessors{i};
        if ~isempty(preds)
            t_en = max(timestamps(preds));
            h = h + beta * max(0, t_en - timestamps(i));
        end
    end
end
```

Bibliography

[1] Wil van der Aalst. *Process Mining: Data Science in Action*. Berlin: Springer, 2016. ISBN: 978-3-662-49850-7, 978-3-662-49851-4 (cit. on p. 1).

[2] Sander J. J. Leemans, Anja F. Syring, and Wil M. P. van der Aalst. «Earth Movers' Stochastic Conformance Checking». In: *Business Process Management Forum*. Ed. by Thomas Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling. Cham: Springer Int. Publishing, 2019, pp. 127–143. ISBN: 978-3-030-26643-1 (cit. on pp. 2, 3).

[3] Artem Polyvyanyy, Alistair Moffat, and Luciano Garc'ia-Banuelos. «An Entropic Relevance Measure for Stochastic Conformance Checking in Process Mining». In: *2020 2nd Int. Conf. on Process Mining (ICPM)* (2020), pp. 97–104. URL: https://api.semanticscholar.org/CorpusID:220646747 (cit. on p. 2).

[4] Adam Burke, Sander J. J. Leemans, and Moe Thandar Wynn. «Stochastic Process Discovery by Weight Estimation». In: *ICPM 2020 Int. Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers*. Vol. 406. Lecture Notes in Business Information Processing. Springer, 2020, pp. 260–272. DOI: 10.1007/978-3-030-72693-5_20. URL: https://doi.org/10.1007/978-3-030-72693-5%5C_20 (cit. on p. 2).

[5] Adam Burke, Sander J. J. Leemans, and Moe Thandar Wynn. «Discovering Stochastic Process Models by Reduction and Abstraction». In: *42nd Int. Conf., PETRI NETS June 23-25, 2021, Proceedings*. Vol. 12734. LNCS. Springer, 2021, pp. 312–336. DOI: 10.1007/978-3-030-76983-3_16. URL: https://doi.org/10.1007/978-3-030-76983-3%5C_16 (cit. on p. 2).

[6] Sander J. J. Leemans, Tian Li, Marco Montali, and Artem Polyvyanyy. «Stochastic Process Discovery: Can It Be Done Optimally?» In: *Advanced Information Systems Engineering: 36th Int. Conf., CAiSE 2024, Limassol, Cyprus, June 3–7, 2024, Proceedings*. Limassol, Cyprus: Springer-Verlag, 2024, pp. 36–52. ISBN: 978-3-031-61056-1. DOI: 10.1007/978-3-031-61057-8_3. URL: https://doi.org/10.1007/978-3-031-61057-8_3 (cit. on p. 2).

[7] Tobias Brockhoff, Merih Seran Uysal, and Wil M.P. Van Der Aalst. «Wasserstein Weight Estimation for Stochastic Petri Nets». In: *2024 6th International Conference on Process Mining (ICPM)*. 2024, pp. 81–88. DOI: 10.1109/ICPM63005.2024.10680664 (cit. on p. 2).

[8] Pierre Cry, András Horváth, Paolo Ballarini, and Pascale Le Gall. «A Framework for Opt imisation Based Stochastic Process Discovery». In: *Quantitative Evaluation of Systems (QEST) and Formal Modeling and Analysis of Timed Systems (FORMATS)*. Vol. 14996. LNCS. Cham: Springer Nature Switzerland, 2024, pp. 34–51. ISBN: 978-3-031-68416-6 (cit. on p. 2).

[9] 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. URL: https://doi.org/10.1007/978-3-319-99414-7 (cit. on pp. 2, 12).

[10] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. «Conformance Checking Using Cost-Based Fitness Analysis». In: *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*. IEEE, 2011, pp. 55–64. DOI: 10.1109/EDOC.2011.12 (cit. on pp. 3, 19, 20, 32).

[11] Tian Li, Artem Polyvyanyy, and Sander Leemans. «Stochastic Alignments: Matching an Observed Trace to Stochastic Process Models». In: *Business Process Management*. 2025. DOI: 10.1007/978-3-032-02867-9_12 (cit. on pp. 3, 53).

[12] Emilio Incerto, Andrea Vandin, and Sima Sarv Ahrabi. «Stochastic conformance checking based on variable-length Markov chains». In: *Information Systems* 133 (2025), p. 102561. ISSN: 0306-4379. DOI: 10.1016/j.is.2025.102561 (cit. on p. 3).

[13] Tian Li, Artem Polyvyanyy, and Sander J. J. Leemans. *Stochastic Alignments: Matching an Observed Trace to Stochastic Process Models*. 2025. arXiv: 2507.06472 [cs.FL]. URL: https://arxiv.org/abs/2507.06472 (cit. on p. 3).

[14] Theodore J. Perkins. «Maximum likelihood trajectories for continuous-time Markov chains». In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta. Curran Associates, Inc., 2009, pp. 1–9 (cit. on pp. 3, 26).

[15] Y. Grinberg and T. J. Perkins. «State Sequence Analysis in Hidden Markov Models». In: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. Amsterdam, The Netherlands, 2015, pp. 336–344. URL: http://auai.org/uai2015/proceedings/papers/167.pdf (cit. on p. 3).

[16] P. Levin, J. Lefebvre, and T. J. Perkins. «What do molecules do when we are not looking? State sequence analysis for stochastic chemical systems». In: *Journal of The Royal Society Interface* 9.77 (2012), pp. 3411–3425. DOI: 10.1098/rsif.2012.0633. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsif.2012.0633 (cit. on p. 3).

[17] Thomas Chatain and Neha Rino. «Timed Alignments». In: *2022 4th International Conference on Process Mining (ICPM)*. hal-03703712. 2022, pp. 112–119. DOI: 10.1109/ICPM57379.2022.9980687 (cit. on pp. 3, 14, 15).

[18] F. Richter, J. Sontheim, L. Zellner, and T. Seidl. «TADE: Stochastic Conformance Checking Using Temporal Activity Density Estimation». In: *Business Process Management - BPM 2020*. Vol. 12168. Lecture Notes in Computer Science. Springer, 2020, pp. 223–240. DOI: 10.1007/978-3-030-58666-9_13 (cit. on p. 4).

[19] Wil van der Aalst. *Process Mining: Data Science in Action*. 2nd. Springer Publishing Company, Incorporated, 2016. ISBN: 3662498502 (cit. on pp. 6, 7, 10).

[20] van der Aalst and al. «Workflow Mining: Discovering Process Models from Event Logs». In: *IEEE Transactions on Knowledge and Data Engineering*. 2004 (cit. on p. 6).

[21] Tadao Murata. «Petri nets: Properties, analysis and applications». In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580. DOI: 10.1109/5.24143 (cit. on p. 7).

[22] E. Best. «Structure Theory of Petri Nets: the Free Choice Hiatus». In: *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part I*. London, UK: Springer-Verlag, 1987, pp. 168–205 (cit. on p. 9).

[23] J. Desel and J. Esparza. *Free Choice Petri nets*. Cambridge University Press, 1995 (cit. on p. 9).

[24] Eike Best, Raymond Devillers, and Evgeny Erofeev. «A New Property of Choice-Free Petri Net Systems». In: *Application and Theory of Petri Nets and Concurrency: 41st International Conference, PETRI NETS 2020, Paris, France, June 24–25, 2020, Proceedings*. Vol. 12152. Lecture Notes in Computer Science. Springer, 2020, pp. 89–108. DOI: 10.1007/978-3-030-51831-8_5 (cit. on p. 9).

[25] Jiacun Wang. *Timed Petri Nets: Theory and Application*. The International Series on Discrete Event Dynamic Systems. New York, NY: Springer, 1998. ISBN: 978-1-4613-7767-8. DOI: 10.1007/978-1-4615-5537-7 (cit. on p. 9).

[26] Marco Ajmone Marsan. «Stochastic Petri nets: An elementary introduction». In: *Advances in Petri Nets 1989. APN 1988*. Ed. by Grzegorz Rozenberg. Vol. 424. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1990, pp. 1–29. DOI: 10.1007/3-540-52494-0_23 (cit. on pp. 10, 20, 33).

[27] S. Natkin. «Les Réseaux de Petri Stochastiques et leur Application à l'Évaluation des Systèmes Informatiques». Thèse de Docteur Ingénieur. Paris, France: Conservatoire National des Arts et Métiers (CNAM), 1980 (cit. on p. 10).

[28] Michael K. Molloy. «On the integration of delay and throughput measures in distributed processing models». In: 1981. URL: https://api.semanticscholar.org/CorpusID:58721967 (cit. on p. 10).

[29] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. «Split miner: automated discovery of accurate and simple business process models from event logs». In: *Knowledge and Information Systems* 59 (May 2019). DOI: 10.1007/s10115-018-1214-x (cit. on p. 12).

[30] A. Adriansyah. «Aligning observed and modeled behavior». English. Phd Thesis 1 (Research TU/e / Graduation TU/e). Mathematics and Computer Science, 2014. ISBN: 978-90-386-3574-3. DOI: 10.6100/IR770080 (cit. on p. 12).

[31] Arya Adriansyah. *Aligning Observed and Modeled Behavior: An Approach to Log-Based Conformance Checking*. Berlin, Heidelberg: Springer, 2014. DOI: 10.1007/978-3-642-39873-4 (cit. on p. 13).

[32] M. Natkin and J. Molloy. «A General Theory of Stochastic Petri Nets». In: *Proceedings of the 3rd International Conference on Application and Theory of Petri Nets*. Lecture Notes in Computer Science. Springer, 1982 (cit. on p. 19).

[33] George Casella and Roger L Berger. *Statistical inference*. Vol. 2. Duxbury Pacific Grove, CA, 2002 (cit. on p. 21).

[34] Bernard Kolman and Robert E. Beck. *Elementary Linear Programming with Applications*. Amsterdam: Elsevier Science & Technology Books, 1995. ISBN: 9780124179103 (cit. on p. 22).

[35] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge: Cambridge University Press, 2004 (cit. on p. 22).

[36] José Manuel Colom and Jörg Desel, eds. *Application and Theory of Petri Nets and Concurrency - 34th Int. Conf., PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*. Vol. 7927. LNCS. Springer, 2013. ISBN: 978-3-642-38696-1 (cit. on p. 33).

[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». In: *CoRR* abs/1905.06169 (2019). arXiv: 1905.06169. URL: http://arxiv.org/abs/1905.06169 (cit. on p. 33).

[38] Sander J.J. Leemans, Fabrizio Maria Maggi, and Marco Montali. «Enjoy the silence: Analysis of stochastic Petri nets with silent transitions». In: *Information Systems* 124 (2024), p. 102383. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2024.102383. URL: https://www.sciencedirect.com/science/article/pii/S0306437924000413 (cit. on p. 33).

[39] Harleen Kaur, Jan Mendling, Christoffer Rubensson, and Timotheus Kampik. *Timeline-based Process Discovery*. 2023. arXiv: 2401.04114 [cs.HC]. URL: https://arxiv.org/abs/2401.04114 (cit. on p. 34).

[40] Dominique Sommers, Natalia Sidorova, and Boudewijn F. van Dongen. «In system alignments we trust! Explainable alignments via projections». In: *Inf. Syst.* 136 (2026), p. 102631. DOI: 10.1016/J.IS.2025.102631. URL: https://doi.org/10.1016/j.is.2025.102631 (cit. on p. 39).

[41] Volker Diekert. *The Book of Traces*. Ed. by Grzegorz Rozenberg. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1995. ISBN: 9810220588 (cit. on p. 39).

[42] Michael Pinedo. *Elements of Scheduling*. Springer, 2016 (cit. on p. 48).

[43] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Jan. 2022. ISBN: 978-3-031-05920-9. DOI: 10.1007/978-3-031-05921-6 (cit. on p. 48).

[44] J. K. Lenstra and A. H. G. Rinnooy Kan. «Complexity of Scheduling under Precedence Constraints». In: *Operations Research* 26.1 (1978), pp. 22–35. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/169889 (visited on 12/10/2025) (cit. on p. 48).

[45] Arthur B. Kahn. «Topological sorting of large networks». In: *Communications of the ACM* 5.11 (1962), pp. 558–562 (cit. on p. 56).

[46] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd. Section 22.4, pp. 612–618. MIT Press, 2009 (cit. on p. 56).