université PARIS-SACLAY

From Event Logs to Probabilistic Models: Methods for Stochastic Process Discovery

Des journaux d'événements aux modèles probabilistes : méthodes pour la découverte de processus stochastiques

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°573: interfaces: matériaux, systèmes, usages (INTERFACES)
Spécialité de doctorat: Informatique
Graduate School: Sciences de l'ingénierie et des systèmes
Référent: CentraleSupélec

Thèse préparée dans l'unité de recherche **Mathématiques et Informatique pour la Complexité et les Systèmes (Université Paris-Saclay, CentraleSupélec)**, sous la direction de **Paolo BALLARINI**, maître de conférences (HDR) et le co-encadrement de **Pascale LE GALL**, professeure des universités

Thèse soutenue à Paris-Saclay, le 19 décembre 2025, par

Pierre CRY

Composition du jury

Membres du jury avec voix délibérative

Benedikt BOLLIG Research Director, Université Paris-Saclay	Président
Sander LEEMANS Professor, RWTH Aachen University	Rapporteur & Examinateur
Andrea VANDIN Associate Professor, Sant'Anna School for Advanced Studies	Rapporteur & Examinateur
Natalia SIDOROVA Assistant Professor, Eindhoven University of Technology	Examinatrice
Thomas CHATAIN Associate Professor, Université Paris-Saclay	Examinateur
Anne BARROS Professor, Université Paris-Saclay	Examinatrice

THESE DE DOCTORAT
NNT : 2025UPAST149

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre: Des journaux d'événements aux modèles probabilistes : méthodes pour la découverte de processus stochastiques

Mots clés: Minage de Processus, Découverte de processus stochastique, Inférence de modèles, Réseau de Petri, Arbre de processus

Résumé : Les systèmes d'information modernes génèrent en continu de vastes volumes de données d'événements, qui offrent un enregistrement détaillé de la manière dont les processus sont exécutés en pratique. Exploiter ces données est essentiel pour comprendre le comportement des systèmes et étayer des améliorations fondées sur les données. Le minage de processus s'appuie sur cette idée en dérivant des modèles de processus à partir de journaux d'événements, c'est-à-dire des collections structurées d'événements où chacun consigne l'exécution d'une activité au sein d'une instance de processus. Alors que les techniques classiques de découverte se concentrent sur la perspective du flux de contrôle, soit les relations d'ordre et de branchement entre activités, elles négligent généralement la dimension stochastique des processus, à savoir la probabilité avec laquelle différents comportements se produisent. Cet aspect probabiliste est pourtant crucial pour l'analyse de performance, le suivi prédictif et l'aide à la décision. Cette thèse aborde le problème de la découverte de processus stochastiques, qui vise à construire des modèles reproduisant à la fois les relations structurelles entre activités et la distribution de probabilité sur les traces observées. Dans cette optique, nous développons des méthodes nouvelles selon deux axes complémentaires : l'estimation fondée sur l'optimisation et l'inférence statistique. Premièrement, nous introduisons une procédure de dépliage pilotée par le journal permettant le calcul exact du langage stochastique des réseaux de Petri stochastiques. Cette procédure sert de fondement à un cadre d'optimisation s'appuyant sur des mesures info-théoriques et basées sur des distances afin d'aligner les modèles sur les distributions issues des journaux d'événements. Deuxièmement, nous proposons une approche d'inférence fondée sur le calcul bayésien approximatif, qui estime les paramètres de probabilité par échantillonnage et simulation itératifs. Enfin, nous définissons le formalisme des arbres de processus stochastiques, alliant l'interprétabilité des arbres de processus à des sémantiques stochastiques, et proposons des algorithmes pour leur découverte. Des expériences approfondies sur des journaux réels issus des défis Business Process Intelligence (BPI Challenges) et sur des journaux synthétiques mettent en évidence l'efficacité et la scalabilité des approches proposées. Cette thèse pose les bases d'un cadre reproductible pour la découverte et l'analyse des comportements probabilistes dans les processus métier, permettant ainsi une analyse, une prédiction et une aide à la décision plus précises.

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Title: From Event Logs to Probabilistic Models: Methods for Stochastic Process Discovery
Keywords: Process Mining, Stochastic Process Discovery, Model Inference, Petri Net, Process Tree

Abstract: Modern information systems continuously generate vast amounts of event data, which provide a detailed record of how processes are executed in practice. Leveraging such data is essential to gain insights into system behaviour and to support evidence-based improvements. Process mining builds on this idea by deriving process models from event logs, that is, structured collections of events where each event records the execution of an activity within a process instance. While traditional discovery techniques focus on the control-flow perspective, the ordering and branching relations between activities, they usually neglect the stochastic dimension of processes, which concerns the likelihood with which different behaviours occur. This probabilistic aspect is, however, crucial for performance analysis, predictive monitoring, and decision support. This thesis addresses the problem of stochastic process discovery, which aims to construct models that reproduce both the structural relations among activities and the probability distribution over observed traces. Building upon this objective, we develop novel methods along two complementary directions: optimisation-based estimation and statistical inference. First, we introduce a log-driven unfolding procedure that enables the exact computation of the stochastic language of stochastic workflow nets, which serves as the foundation for an optimisation framework leveraging information-theoretic and distance-based measures to align models with event log distributions. Second, we propose an inference approach grounded in approximate Bayesian computation, which estimates probability parameters through iterative sampling and simulation. Finally, we define the formalism of stochastic process trees, combining the interpretability of process trees with stochastic semantics, and provide algorithms for their discovery. Extensive experiments on real-life event logs from the Business Process Intelligence challenges and on synthetic benchmarks demonstrate the effectiveness and scalability of the proposed approaches. This thesis lays the foundations of a reproducible framework for discovering and analysing probabilistic behaviours in business processes, thereby enabling more accurate analysis, prediction, and decision support.

“Je vais essayer de me faire RIRE pour voir jusqu’où je peux supporter la DOULEUR.”

— Didier Cry

Acknowledgements

Je suis notoirement mauvais pour dire merci alors, merci de ne pas m'en tenir rigueur. Si ton nom n'apparaît pas ici, deux explications sont possibles : soit je t'ai involontairement oublié, soit je te déteste.

Je voudrais tout d'abord remercier mes parents et mes grands-parents qui, même s'ils ne sont pas venus et même si je ne le leur dis pas souvent, savent que je les aime et que j'apprécie profondément tout ce qu'ils font pour moi. Un merci tout particulier à Jean, qui m'a mis sur la voie de la réussite et a largement contribué à forger l'homme que je suis aujourd'hui.

Je souhaite ensuite remercier Pascale et Paolo, qui ont été les meilleurs encadrants que j'aurais pu espérer. Ils ont toujours fait preuve d'une grande attention et d'un réel intérêt pour mon travail et mon bien-être, et m'ont offert un accompagnement que peu de jeunes chercheurs ont la chance de connaître. Je remercie également tout particulièrement András Horváth, qui a participé activement à l'ensemble des contributions de ce travail et avec qui c'est un réel plaisir de collaborer.

Je voudrais maintenant remercier Emmanuelle Claeys. Tu as été la première personne avec qui j'ai pu faire de la recherche, et la première à croire en moi dans ce domaine. C'est grâce à ta bienveillance et à ton aide acharnée que j'ai pu commencer cette thèse. Je remercie également Benoît Barbot, qui m'a aussi beaucoup aidé à mes débuts.

Un grand merci à tous mes potes d'Albi et de Narbonne. Les vrais potes sont ceux qui connaissent parfaitement vos points faibles et n'hésitent jamais à vous descendre. Vous avez toujours été là pour m'aider, me vanner, et m'accepter tel que je suis. Merci à Jo, qui est un sale gosse pourri gâté et égoïste ; à Vic, qui est nul à tous les jeux ; à Val, dont les yeux semblent parfois un peu trop proches ; à Quent, dont la relation avec la clope laisse peu de place à l'optimisme ; à Ponsol, spécialiste incontesté de la falsification de bulletins ; à Apo, pour ses goûts horribles, qui préfère Triss à Yennefer et Shakira à Sydney Sweeney ; à Lilian, qui ne reviendra probablement jamais vivre à Paris ; à Plessis, le squatteur d'appartements ; et à Fràcx, le pire joueur de *LoL* de la planète.

Je voudrais également remercier chaleureusement mes amis et collègues du laboratoire, qui ont rendu mes longues heures de bureau un peu plus rythmées. Merci en particulier à mon voisin de bureau Marien, qui rigole toujours à la moindre de mes blagues et a été un véritable soutien pendant la rédaction du manuscrit.

Merci à Troma, qui devrait manger un peu plus et un peu plus vite. Merci à Gabriel que j'aime beaucoup, et à Gabriel que j'aime un peu moins, mais ça va quand même. Merci à Tocard pour ses explications sur les martingales. Merci à Francesco, grâce à qui j'espère un jour courir plus vite et plus longtemps que lui. Merci à Nicoleta pour le magnifique cactus. Merci également à Hakim, Hanna, David, Jérémy, Raffaele, Dimitra, Gérault, Alexandre et Armand.

Je remercie aussi mes collègues de biomaths, que j'adore détester. Merci à Imane pour les cours d'anglais et d'espagnol et d'italien, à Malek pour les tartines arabes, ainsi qu'à Margherita, Manon, Margot et Inès.

Merci beaucoup à Fabienne et Leïla, véritables rayons de soleil de ce laboratoire, qui ont toujours su m'aider dans mes démarches avec beaucoup de gentillesse et de sérieux. Merci à Guillaume, à Romain, à Brice et à Paul pour leurs précieux conseils, le partage de leur expérience et surtout leur capacité à toujours animer les discussions. Enfin, merci à tous les autres membres du laboratoire avec qui c'est toujours un plaisir de discuter et travailler : Céline, Marc, Vincent, Wassila, Anaëlle et Gilles.

Table of Contents

List of Figures

List of Tables

List of Acronyms

PN	Petri Net
WN	Workflow Net
sWN	Stochastic Workflow Net
PT	Process Tree
sPT	Stochastic Process Tree
ABC	Approximate Bayesian Computation
ABC-SMC	Approximate Bayesian Computation with Sequential Monte Carlo
HASL	Hybrid Automata Stochastic Language
LHA	Linear Hybrid Automaton
KLD	Kullback-Leibler divergence
LH	Log-likelihood distance
rEMD	Restricted Earth Mover's Distance
tEMD	Truncated Earth Mover's Distance
uEMSC	Unit Earth Mover's Stochastic Conformance
BPIC	Business Process Intelligence Challenge
WaWE	Wasserstein Weight Estimation
SLPN	Stochastic Labeled Petri Net
PM4PY	Process Mining for Python
PDF	Probability Density Function
CDF	Cumulative Distribution Function

Chapter 1

Introduction

Over time, data has become an increasingly central component of our daily lives, helping shape how we understand, analyse, and improve complex systems. Every event, whether triggered by a deliberate interaction or occurring spontaneously, tells a fragment of a story that can be pieced together and reveals the underlying dynamics of the systems we observe. Reconstructing and analysing these stories is at the heart of modern data-driven science. Process mining [74] is a research field that aims to extract meaningful knowledge from those fragments. It provides methods to collect and structure information, recognise patterns of behaviour, and ultimately build concise and interpretable models that capture the essential characteristics of a process as a whole. From this representation, we can utilise past experiences to predict and improve the future of these systems.

Beyond the structural description of processes, a more subtle and increasingly important challenge of process mining lies in leveraging the stochastic information naturally embedded in data [48]. This probabilistic information reflects the likelihood of observing certain behaviours as the system operates. When correctly modelled, it unveils an entirely new dimension of analysis and can significantly influence the insights drawn by process analysts and domain experts. While the use of stochastic models in process mining is not new, the development of efficient stochastic process discovery approaches capable of issuing models that adequately reproduce the stochastic character of the considered log remains, at this time, a relatively young research branch. It recently gained interest from the community and is being identified as a key research challenge [3]. The difficulty with the discovery of a stochastic process is that the resulting models shall not only be able to reproduce the observed executions (meaning the ordering with which observed events have occurred) but also the likelihood with which they have been observed. Since several probabilistic parameters characterise stochastic process models (and affect the likelihood of the executions they generate), the discovery of a stochastic process entails, among other things, determining the optimal values of these parameters

with respect to a given stochastic conformance criterion. This thesis addresses the problem of extracting a stochastic model that optimally reproduces the stochastic character of the considered systems.

This introductory chapter begins with a detailed presentation of the objectives and scope of process mining, as well as a more thorough introduction to the formalism of processes. We then introduce the problem of discovering stochastic processes, outlining the aims of this thesis and summarising the goals and contributions of each chapter.

1.1 Processes and Process Mining

A business process [44] comprises activities that aim to achieve a specific organisational goal. Understanding business processes is crucial for organisations in several respects: first, to obtain a formal description of processes that are often undocumented, and second, to use the resulting process model to monitor, analyse, and ultimately improve the process under study.

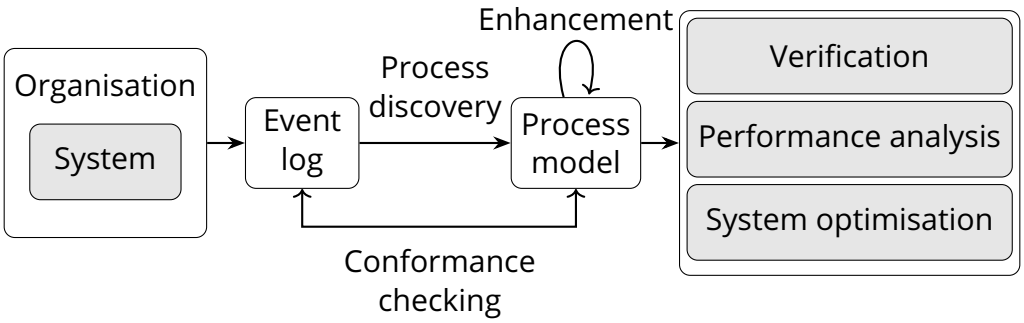


Figure 1.1 – An overview of process mining

At the core of process mining lies the event log, which provides the raw observational data describing how processes unfold and interact within real-world systems. Transforming these logs into meaningful representations that support understanding, diagnosis, and improvement requires rigorous and well-defined methodologies. Process mining addresses this challenge by exploiting those collections of events to uncover the underlying structure and behaviour of the business processes. Process mining techniques are typically classified into three main categories [2]:

- *Process Discovery*, where a process model is automatically generated from an event log without relying on any prior model;
- *Conformance checking*, which compares an event log with an existing process model to assess whether the observed behaviour fits the model or deviates from it. It provides diagnostics that identify non-conforming behaviours or activities and may quantify the level of similarity between the log and the model in terms of multiple metrics, such as fitness or precision [24, 4].

- *Enhancement*, that seeks to refine or extend an existing process model by incorporating additional perspectives extracted from the log. This includes enriching the model with performance indicators (e.g., bottlenecks, waiting times) or resource information and improving its descriptive and predictive quality [75, 74].

As illustrated in Figure 1.1, all these components interact with one another to build the most accurate and insightful model possible. Models can then be exploited in several domains to perform verification, performance analysis, or to serve as a foundation for system optimisation in collaboration with domain experts. For instance, such models can be used in healthcare to optimise patient care pathways, in manufacturing to streamline production processes, in finance to detect fraud and improve transaction handling, or in telecommunications to analyse and enhance customer service processes.

1.1.1 Event logs: a fundamental unit for capturing information in business processes

In practice, an event log [74] is represented as a collection of events, where each event captures an interaction between the system and its environment, typically specifying the activity performed, its timestamp, the resource involved, and additional contextual information. The environment may include both human actors (e.g., employees handling cases, patients undergoing procedures) and automated components such as sensors, machines, or software services. What is interesting when it comes to discovering, evaluating and enhancing the process described by an event log is to consider the subsequent observed executions of the process as a collection of traces, each trace being a sequence of timestamped events that describe the evolution of a single case. Since the same sequence of activities can occur in multiple cases, an event log may contain several instances of identical traces. The resulting trace frequency quantifies how often each unique sequence of actions is observed, thereby reflecting the likelihood of the corresponding behaviour within the process. The set of traces, together with their frequencies, constitutes a language that describes the behaviour recorded in the event log. In the same spirit, by normalising the trace frequencies, we obtain an empirical likelihood distribution over traces, which gives rise to a stochastic language.

For example, consider an artificial event log that describes the flow of patients through various hospital services over a single day. Table 1.1 provides an excerpt of this log, consisting of 18 individual events, one per line. In this scenario, each case corresponds to a patient and records the procedures they underwent. The first event describes an analysis performed in the haematology service, possibly a blood test, by Dr Lefevre on patient LP4721, on July 28th, 2025, at 11:20. By exploring the entire log, we can extract every unique sequence of activities and associate each one with the number of times a particular case executed it. Table 1.2 presents the complete list of sequences in the log and their respective frequencies. For example, if we consider the first sequence, we observe that, on that day, 30 patients followed

exactly this series of activities within the hospital. Empirically, the likelihood of this trace is 0.3, given that it occurs in 30 out of the 100 cases recorded in the log.

Event	Case	Activity	Timestamp	Service	Professional
1	LP4721	Analysis	28/07/2025 11:20	Hematology	Dr.Lefevre
2	HV5689	Registration	28/07/2025 11:28	Emergency	Dr.Morel
3	HV6732	Registration	28/07/2025 11:30	Emergency	Dr.Denis
4	LP4721	Triage	28/07/2025 11:35	Emergency	Dr.Bernard
5	HV5689	Analysis	28/07/2025 11:35	Radiology	Dr.Bernard
6	HV6732	Analysis	28/07/2025 11:40	Hematology	Dr.Lopez
7	HV6732	Analysis	28/07/2025 11:50	Radiology	Dr.Laurent
8	HV5689	Consultation	28/07/2025 11:50	Consult unit	Dr.Bernard
9	LP4721	Consultation	28/07/2025 12:00	Consult unit	Dr.Fontaine
10	HV6732	Consultation	28/07/2025 12:05	Consult unit	Dr.Laurent
11	HV5689	Discharge	28/07/2025 12:10	Emergency	Dr.Morel
12	HV6732	Discharge	28/07/2025 12:15	Emergency	Dr.Denis
13	LP4721	Admission	28/07/2025 12:20	Emergency	Dr.Morel
14	LP8391	Registration	28/07/2025 12:25	Emergency	Dr.Denis
15	LP8391	Analysis	28/07/2025 12:30	Hematology	Dr.Lefevre
16	LP8391	Analysis	28/07/2025 12:40	Radiology	Dr.Bernard
17	LP8391	Consultation	28/07/2025 12:50	Consult unit	Dr.Fontaine
18	LP8391	Discharge	28/07/2025 13:00	Emergency	Dr.Morel
...	...	...	...	...	...

Table 1.1 – Chronologically ordered event log of hospital cases

1.1.2 Process model: a compact and exploitable representation of business processes

Depending on the chosen approach and objective, the resulting process model derived from an event log can take various forms, ranging from mathematical objects, such as transition systems like automata or Petri nets [41], to more comprehensive expert graphical representations, like UML diagrams [68]. They can also take the form of a declarative constraint over activity relations [56]. Regardless of their formalism, all process models share the same fundamental goal: to provide a compact, structured, and interpretable representation of the behaviour observed in the process. They describe how log activities are related to one another, thereby capturing both the control-flow and, in some cases, the data and resource perspectives of the process.

In essence, Petri nets constitute a powerful process modelling formalism, offering particular expressiveness in capturing conflict and concurrency [1]. They consist of a set of transitions, labelled with activities from the event log, whose firing represents the occurrence of the corresponding activity within the system. Surrounding the transitions are places which hold tokens and are connected to transitions via arcs. Together, places, transitions, and arcs define the causal and conditional relationships that govern when and how transitions can fire, thereby capturing every

Sequence of activities	Frequency	Likelihood
Registration → Analysis → Consultation → Discharge	30	0.3
Triage → Analysis → Consultation → Discharge	25	0.25
Analysis → Triage → Consultation → Admission	20	0.2
Registration → Analysis → Analysis → Consultation → Discharge	15	0.15
Analysis → Registration → Analysis → Consultation → Admission	10	0.1

Table 1.2 – Abstracted sequences of activities for hospital cases

dynamic process behaviour that can be contained. The exact definition and semantics of Petri nets are given in Chapter 2.

Figure 1.2 depicts a Petri net that compactly models all the relations observed in the sequence of activities shown in Table 1.2. The model captures, for instance, that the activities “Registration” and “Triage” cannot occur together in the same process execution, that the “Analysis” activity may occur multiple times, that there are several possible orders in which “Registration”, “Triage”, and “Analysis” can take place, and that “Consultation” always appears at the same position in every execution of the process.

The structure of Petri nets is naturally defined in a way that can be extended to stochastic dimensions. When transitions conflict, assigning weight parameters to them enables the specification of their likelihood. By combining the probabilities of the transitions involved in a particular execution, one can derive the likelihood of the corresponding behaviour.

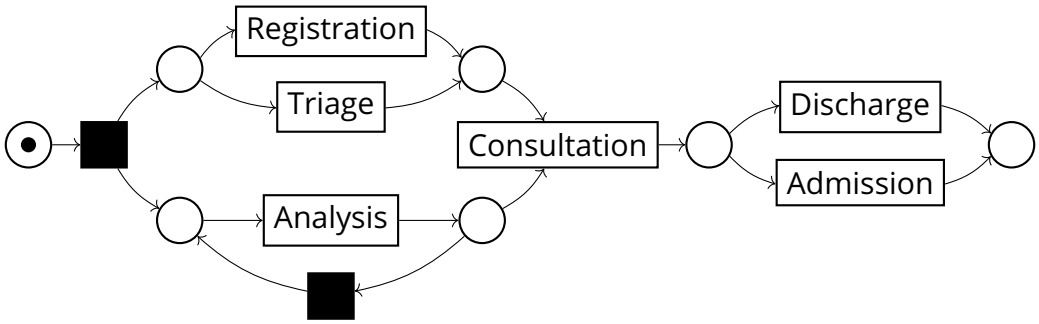


Figure 1.2 – A Petri net describing the process of hospital cases

1.2 The Stochastic Process Discovery Problem

Over the years, a wide variety of process discovery algorithms have been developed, all sharing the common goal of producing readable and exploitable models and capturing as faithfully as possible the behaviour expressed in the event logs.

1.2.1 Discovering procedural and declarative models

Depending on the approach and the context, researchers in the process mining community aim to discover models of different natures [9].

Procedural approaches explicitly specify the control-flow aspect of processes and produce models such as Petri nets [65], Business Process Model and Notation (BPMN) diagrams [87], directly-follows graph [47] or process trees [49]. Well-known process discovery algorithms in this category include the α-algorithm [71], the Heuristic Miner [86], the Inductive Miner [49], the Split Miner [8], and Integer Linear Programming (ILP)-based approaches [76]. Conceptually, such algorithms aim to extract from an event log the control-flow relations that characterise the observed executions. Four fundamental types of relationships between log events are sufficient to capture any behaviour:

- *Sequence*, when one activity must always follow another in the observed order.
- *Conflict*, when only one activity from a set of alternatives can occur at a given point, i.e., the occurrence of one activity excludes the others in the same process instance.
- *Concurrency*, when activities can occur independently or in parallel. This is typically observed when the same group of activities appears in varying timestamp orders across different instances.
- *Loop*, when an activity or group of activities may repeat within the same process instance.

These relations serve as the basis for constructing a process model that encodes them according to the model's formal semantics.

In contrast, *declarative approaches* describe processes not by explicitly modelling their control-flow, but by specifying a set of constraints that any valid execution must satisfy. They characterise the process in terms of rules such as "activity A must eventually be followed by activity B" or "activity C and D cannot occur together in the same trace. This paradigm is more flexible, as it allows managing both the number and the granularity of the rules depending on the desired level of conformance, making it particularly suitable for processes that are highly variable and weakly structured. For such processes, a control-flow approach would have to produce too many relations to keep the model readable. Representative techniques include the Declare Miner [56] or MINERful [27].

1.2.2 Discovering stochastic models

Although proven effective in many respects, "standard" discovery approaches are limited by an inherent weakness, as they do not consider a valuable source of information in the logs, namely, trace frequency. This may severely undermine the ability to improve the modelled business, as the discovered models cannot reproduce the likelihood of the observed behaviours. This gives rise to *stochastic process mining*, whose goal is to discover stochastic models that not only capture the control-flow or constraints of a process, but also reproduce the likelihood of its observed behaviour, as in stochastic extensions of Petri nets [19] and probabilistic process constraints [6]. This can be achieved either *indirectly*, i.e., using a standard miner to obtain a non-stochastic model and then, based on the trace frequencies, devising adequate stochastic parameters to convert it into a stochastic model, or *directly*, i.e., by devising a stochastic model straight from the log. In general, these strategies extend the formalism of existing process models with a stochastic dimension, where additional parameters quantify the relative likelihood of choices, repetitions, or concurrent executions. Once identified, these parameters can be studied and optimised to obtain a model that faithfully mirrors both the structure and the probabilistic dynamics of the process.

To the best of our knowledge, research on discovering stochastic process models from sequential data, such as traces, can be broadly divided into three methodological categories, as outlined in the PAutomaC competition benchmark [82] and in more recent process discovery work [22]: Bayesian inference, state merging, and parameter estimation. The PAutomaC competition [82] was a large-scale benchmark for comparing algorithms that learn probabilistic automata from sequential data. Among the most effective approaches were Collapsed Gibbs Sampling [69] (Bayesian inference), Alergia [25] (state merging), and the Baum-Welch algorithm [12] (parameter estimation). In the context of modelling event logs with stochastic models, the focus shifts from learning stochastic automata to adapting the three strategies so that they directly address the estimation of stochastic parameters within process models.

- *Parameter estimation.* These approaches follow an indirect strategy, assuming that the control-flow structure of the model is already given, and focus on inferring its stochastic parameters from the log. Probabilities are typically assigned to branching or looping constructs based on their observed relative frequencies. This can be achieved through well-defined heuristics and log-driven estimators [21], or by means of optimisation procedures [45, 17]. While efficient, this strategy critically depends on the correctness of the underlying structural model.

- *Bayesian inference.* In this approach, model parameters are treated as random variables, and their posterior distributions are estimated given the observed data collected during the discovery procedure. This provides a principled framework to capture uncertainty and variability in the process, often relying on sampling-based techniques such as Markov Chain Monte Carlo.

Nevertheless, these methods are computationally demanding and, to date, have been applied only to a limited extent in process mining (e.g., [39]).

- *State merging.* This family of methods starts from an extremely detailed representation of the log, where every trace is explicitly represented. This initial structure is usually too specific, tends to overfit the data, and results in enormous complexity. The algorithm then progressively merges states that behave similarly, producing a more compact and general model that also captures probabilities. This strategy has recently been adapted for use in process mining, for instance, in the *Toothpaste* algorithm [22].

1.3 Research Aims and Problematics

The core problem addressed in this thesis is how to automatically discover process models that are not only structurally correct but also stochastically accurate. This entails defining suitable modelling formalisms, developing efficient algorithms to extract both structure and probabilities from stochastic models, designing techniques to evaluate conformance, and finally providing procedures to discover such models in practice.

1.3.1 Motivation

As discussed earlier, the stochastic aspect of a process can reveal critical information about the system's functioning under study. The interpretation of this information may vary depending on the application domain, but overall, stochastic data highlights which behaviours are most probable and which are less frequent or exceptional. As a first level of analysis, this can help assess whether the most probable traces correspond to the expected and desirable behaviour of the system and its environment, and whether the least probable traces reflect rare but valid exceptions, or possibly noise or anomalies in the log [15, 40]. At a higher level of analysis, this information can help isolate parts of the system, such as human or material resources, that are either overused, underutilised, or entirely unused [35].

Interpreting the empirical distribution. From Table 1.2, the abstracted log contains 5 unique traces with frequencies that induce an empirical distribution:

$$(30,\ 25,\ 20,\ 15,\ 10) \quad \Rightarrow \quad \hat{\mathbb{P}}(\sigma) = \tfrac{1}{100}\,(0.30,\ 0.25,\ 0.20,\ 0.15,\ 0.10)\,.$$

The two most probable paths are:

$$\sigma_1 = \langle \text{Registration}, \text{Analysis}, \text{Consultation}, \text{Discharge} \rangle \quad (30\%),$$
$$\sigma_2 = \langle \text{Triage}, \text{Analysis}, \text{Consultation}, \text{Discharge} \rangle \quad (25\%),$$

and the least frequent paths are:

$$\sigma_3 = \langle \text{Analysis}, \text{Triage}, \text{Consultation}, \text{Admission} \rangle \quad (20\%),$$
$$\sigma_4 = \langle \text{Registration}, \text{Analysis}, \text{Analysis}, \text{Consultation}, \text{Discharge} \rangle \quad (15\%),$$
$$\sigma_5 = \langle \text{Analysis}, \text{Registration}, \text{Analysis}, \text{Consultation}, \text{Admission} \rangle \quad (10\%).$$

Here, σ_1 and σ_2 reflect the *expected behaviour* in which a patient arrives via triage or registration, undergoes analysis and consultation, and is discharged without complications. In contrast, σ_4 exhibits rework (a repeated *Analysis*), while σ_3 and σ_5 show atypical orderings (e.g., *Analysis* preceding *Triage* in σ_3, and *Registration* after *Analysis* in σ_5) that end in *Admission*. These traces are prime candidates for root-cause analysis (valid exceptions vs. noise). A stochastic model reproducing these observations enables quantitative conformance checks, hypothesis testing about rework and atypical orderings, and data-driven assessment of resource allocation (over-/under-utilisation).

Probability distributions over identical traces. To illustrate the importance of probabilities, consider two event logs E_1 and E_2 that both generate exactly the same set of traces $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$, but assign different likelihoods to them. In the first log E_1, the probabilities are close to the empirical distribution observed in Table 1.2:

$$\hat{\mathbb{P}}_{E_1} = (0.30,\ 0.25,\ 0.20,\ 0.15,\ 0.10)\,.$$

In contrast, the second log E_2 yields a mirrored distribution, reversing the relative likelihoods of the traces:

$$\hat{\mathbb{P}}_{E_2} = (0.10,\ 0.15,\ 0.20,\ 0.25,\ 0.30)\,.$$

In both cases, the support of the language is identical, but the interpretation changes drastically. E_1 reflects the expected process behaviour, in which patients are most likely discharged without complications. E_2, however, suggests that the most likely outcomes are those involving rework or atypical orderings ending in admission. This example shows that without considering the stochastic dimension, two processes could be judged equally fitting in terms of control-flow, while in reality they convey very different insights about the process dynamics. Applying a non-stochastic discovery algorithm to both logs would yield the same process model. While such a model could still provide valuable insights into the functioning of the two hospitals, it would hide the critical fact that traces of hospital E_2 corresponding to the expected behaviour are among the least frequent. This observation in a stochastic model can help detect malfunctions and identify their causes, as well as provide insight into finding solutions to these malfunctions.

1.3.2 Research questions

This thesis addresses key challenges in discovering stochastic process models that accurately and efficiently reproduce observed behaviours in an indirect approach to parameter estimation and Bayesian inference strategies. To address these questions, we propose methods that combine model analysis, optimisation techniques, and statistical inference to uncover the probabilistic nature of processes from event data.

(Q1) Given a stochastic process model, can we compute, exactly or approximately, the traces it can produce together with their probabilities?

Many applications require fast and accurate extraction of the behaviour of a discovered process model. The most immediate use is to assess the model's conformance against the original event log. A second motivation is model improvement, where one aims to refine the discovered model by increasing its alignment with the observed log, extending coverage to legitimate but missing traces, or enriching the model with additional dimensions (e.g., probabilistic or temporal information). Beyond conformance and improvement, extracting the probabilistic behaviour of a model is also a prerequisite for tasks such as performance prediction, resource analysis, or risk assessment. In all these cases, we need efficient procedures to compute the model's *probabilistic behaviour*, that is, the probabilities it assigns to sets of traces. Depending on the setting, computations may be exact, when the discoverer requires complete precision, or approximate, when the discoverer must quickly test multiple models. We explore both possibilities:

- *Exact methods.* We define a log-driven exploration and unfolding of the stochastic model's state space. The procedure systematically identifies and explores the branches of the model that can generate the observed log traces, while keeping track of the cumulative probabilities associated with each model path.
- *Approximate methods.* We define an approximate approach that relies on statistical model checking via simulation, guided by hybrid automaton logics. The approach consists of sampling a large number of executions from the model and using them to approximate its stochastic behaviour.

Both methods will be integrated into a specific iterative stochastic discovery framework, tailored to the chosen strategy.

(Q2) Given a parametric stochastic model, can we design a procedure to identify the optimal (with respect to a chosen stochastic conformance measure) stochastic parameters that match the observed behaviour stored in the log?

In an indirect approach to stochastic process discovery, we start from a control-flow model and calibrate stochastic parameters (e.g., branching probabilities or transition weights) so that the model's induced probabilistic behaviour aligns with the empirical distribution observed in the event log. We investigate two strategies:

- *Optimisation-based parameter estimation.* We define an objective function to quantify the divergence between the model and the log based on a stochastic conformance checking metric, and optimisation techniques are used to minimise this divergence. Both gradient-based solvers and gradient-free solvers are considered.
- *Simulation-based Bayesian inference.* We adopt an Approximate Bayesian Computation with Sequential Monte Carlo (ABC-SMC) procedure to construct posterior distributions over the stochastic parameters of the model. To this end, parameter values are iteratively sampled by simulating model executions and retaining those that achieve a level of conformance sufficiently close to the log, under a progressively decreasing tolerance schedule.

Both strategies provide complementary ways of calibrating stochastic process models: optimisation-based approaches aim for efficient point estimates of the parameters, whereas Bayesian inference delivers complete posterior distributions at a higher computational cost.

(Q3) Can we extend hierarchical process models, such as process trees, with stochastic semantics that faithfully capture the probabilistic behaviour observed in event logs?

Beyond calibrating stochastic parameters on a control-flow Petri net, we seek an alternative modelling formalism that is both *discoverable* from logs and *exploitable* for quantitative analysis. Classical stochastic Petri nets offer a rigorous semantics, but they often suffer from structural ambiguities and difficulties in parameter identifiability when applied to discovery. To overcome these issues, we posit that *stochastic process trees* (sPTs) constitute a suitable representation. An sPT extends the well-established process tree formalism [49] with probabilistic annotations on control-flow operators such as sequence, choice, parallel and loop. Each operator is equipped with parameters that define the likelihood of alternative paths, the distribution of concurrent interleavings, or the expected number of loop iterations. The semantics of sPTs is compositional, meaning that the stochastic behaviour of a complex model is entirely determined by the behaviour of its subcomponents, combined according to their operator. Compared with other stochastic models, sPTs reduce structural ambiguity (avoiding multiple distinct models that generate the same behaviour) and improve parameter identifiability by making probabilistic choices explicit in the model's syntax. They thus provide a unified framework where the structure can be mined directly from logs, the parameters can be calibrated, and the resulting model can be analysed both qualitatively with respect to the control-flow and quantitatively with respect to the probabilities.

1.4 Contributions

Chapter 2. We introduce the notations and formal definitions of the mathematical objects used throughout the thesis. We then present the notion of a language, a central concept for expressing process behaviour compactly and rigorously. Finally, we define the semantics of process models that will serve as the basis for representing and analysing such behaviours. This chapter also provides the background and positions the contributions of this thesis within the broader state of the art, with a focus on stochastic process discovery. It also introduces key existing works and methodologies that will be revisited in the subsequent contribution chapters.

Chapter 3. The first contribution of this thesis is a novel procedure for extracting both the stochastic behaviour and the control-flow structure of a stochastic process model. This method addresses the research question (Q1) by deriving the exact probabilistic distribution over the process instances generated by the model that are consistent with the log, thereby providing a foundation for subsequent optimisation and analysis tasks.

Chapter 4. Building upon the behaviour extraction procedure introduced in the previous chapter, we demonstrate how it can be used to discover an optimised stochastic process model that closely matches the stochastic behaviour observed in an event log, thereby providing an answer to research question (Q2). This is achieved by leveraging well-known optimisation strategies that minimise a distance function between the model and the log.

Chapter 5. The third contribution also addresses, at the same time, research questions (Q1) and (Q2), but approaches it from a simulation-based Bayesian statistical perspective. Here, the parameter functions that govern the model's stochastic behaviour are iteratively approximated by performing a series of simulations. Unlike classical optimisation-based approaches, this strategy scales more effectively to larger and more complex models, while also providing richer insights into the stochastic dynamics of the studied process.

Chapter 6. The final contribution introduces an extension of the process tree modelling formalism itself. It provides an answer to the research question (Q3), aiming for a more compact and less ambiguous representation than the models employed in the previous chapters. This new framework supports a more effective and scalable optimisation procedure than traditional models commonly adopted in process mining.

1.4.1 List of papers

- Publication in an international peer-reviewed journal:
  - **Pierre Cry**, András Horváth, Paolo Ballarini, Pascale Le Gall. "An efficient stochastic process discovery framework based on optimization". In: *Journal on Software Tools for Technology Transfer (STTT)*.
- Publications in international peer-reviewed conferences:
  - **Pierre Cry**, András Horváth, Paolo Ballarini. "Stochastic Process Trees: A Formal Framework for Stochastic Process Discovery". In: *International Conference on Process Mining (ICPM)*, Oct 2025, Montevideo, Uruguay.

  [28] **Pierre Cry**, Paolo Ballarini, András Horváth, Pascale Le Gall. "Statistical Bayesian Inference for Stochastic Process Discovery". In: *International Conference on Quantitative Evaluation of SysTems (QEST)*, Aug 2025, Aarhus, Denmark. HAL Id: hal-05134848

  [11] Paolo Ballarini, Andras Horvath, **Pierre Cry**. "Probabilistic Process Discovery with Stochastic Process Trees". In: *Conference on Performance Evaluation and Optimization of Complex Systems*, Dec 2024, Milan, Italy. HAL Id: hal-05021584

  [29] **Pierre Cry**, Paolo Ballarini, András Horváth, Pascale Le Gall. "A framework for optimisation based stochastic process discovery". In: *Proceedings of the International Conference on Quantitative Evaluation of Systems and Formal Modelling and Analysis of Timed Systems*, Sep 2024, Calgary (Alberta), Canada. doi: 10.1007/978-3-031-68416-6_3

- Publication in a national peer-reviewed conference:

  [32] **Pierre Cry**. “Découverte de processus probabiliste avec des arbres de processus stochastiques (Résumés longs)”. In : *Journées Approches Formelles dans l’Assistance au Développement du Logiciel (AFADL 2025). Approches Formelles dans l’Assistance au Développement du Logiciel*, Jun 2025, Pau, France. HAL Id : hal-05106227

1.4.2 ProDiSt: A Tool for Process Discovery by Stochastic Approaches

To support our contributions and provide a resource to the community, we have made available all the methods, algorithms, data, and experimental results through the `ProDiSt` tool, which is freely accessible at `https://pierrecry98.github.io/tool.html`. `ProDiSt` is a Python-based software platform that integrates a family of approaches for addressing the stochastic process discovery problem.

The current version of `ProDiSt` offers three main indirect stochastic process discovery functionalities, which correspond to the contributions presented in this thesis:

1. Discovery of optimal parameters for a stochastic workflow net (sWN) through the exact computation of its stochastic language [29] (see Chapter 3 and Chapter 4).
2. Inference of the posterior distribution of the optimal weight parameters of a stochastic workflow net via likelihood-free Bayesian estimation [29] (see Chapter 5).
3. Discovery of optimal parameters for a stochastic process tree (sPT) model by approximating its stochastic language [11] (see Chapter 6).

The code supports the import of event logs in the `.xes` format [34] and provides an object-oriented implementation of Petri nets and process trees, along with their stochastic extensions, based on the `PM4Py` package definitions [14].

Chapter 2

Background

In this chapter, we introduce the basic elements of standard control-flow process mining (Section 2.3) and its stochastic extension (Section 2.4). Specifically, in Section 2.3, we provide an overview of commonly used control-flow modelling formalisms (i.e., workflow nets and process trees), outline the principles of conformance checking, and review relevant process discovery methods. In Section 2.4, we instead review the basics of stochastic process mining, starting with the introduction of the stochastic workflow net modelling class, then the stochastic conformance criteria, and eventually surveying related works in the stochastic process discovery domain. Finally, in Section 2.5, we introduce the basics of two statistical frameworks: the statistical model checking method based on Hybrid Automata Stochastic Logic (HASL) and the Approximate Bayesian Computation (ABC) parameter inference approach, which the material presented in Chapter 5 relies upon.

2.1 Mathematical notations

We recall basic mathematical definitions and notations used throughout the manuscript.

A finite *set* S is a collection of distinct objects listed within curly brackets, e.g., $\{a, b, c\}$. The set with no element is called the empty set, denoted $\emptyset$. The cardinality of a finite set S, written $|S|$, is the number of elements it contains. Standard set operations include intersection $\cap$, union $\cup$, subset inclusion $\subseteq$, and set difference $\setminus$. For example, $S_1 = \{a, b, c, d, e\}$ is a set with $|S_1| = 5$.

A finite *sequence* from set S, also called a *word*, is an ordered collection of elements of S, possibly containing repetitions. Given a finite set S, we write S^* for the set of all finite sequences over S. Formally, a finite sequence $\sigma \in S^*$ of length n is a function $\{1, \ldots, n\} \to S$. We use angle brackets $\langle\ \rangle$ to denote sequences,

e.g., $\sigma_1 = \langle a, c, c, d, b, e \rangle \in \{a, b, c, d, e\}^*$. The empty sequence is denoted ε. The length of a sequence σ, written $|\sigma|$, is the number of elements in the sequence, e.g., $|\langle a, c, c, d, b, e \rangle| = 6$, $|\epsilon| = 0$.

A *multiset* M is a generalisation of a set in which elements may appear multiple times. Formally, a multiset over a set S is a function $M \colon S \to \mathbb{N}$ that assigns to each element of S a multiplicity, i.e., the number of times the element appears. We use square brackets $[\,]$ to denote multisets. In contrast, for simplicity, we denote multiplicities through a superscript notation. When the multiplicity of an element is equal to one, the superscript is omitted in the representation. For example $M_1 = [a^3, b, c^2, d^3]$ is a multiset over $S = \{a, b, c, d, e\}$ with multiplicities $M_1(a) = 3$, $M_1(b) = 1$, $M_1(c) = 2$ and $M_1(d) = 3$. By convention, the multiplicity of an element not present in M is defined as 0. The support $supp(M) \subseteq S$ of a multiset M is the set of elements with positive multiplicity in M, e.g. $Supp([a^3, b, c^2, d^3]) = \{a, b, c, d\}$. The set of all finite multisets over S is denoted by $\mathbb{B}(S)$. Formally:

$$\mathbb{B}(S) = \{M \mid M \colon S \to \mathbb{N}, supp(M) \text{ is finite}\}$$

The cardinality $|M|$ is the total number of elements (counting repetitions), while the unique cardinality $||M||$ is the number of distinct elements (i.e., the cardinality of $Supp(M)$). For example, the cardinality of $M_1 = [a^3, b, c^2, d^3]$ is $|M_1| = 9$ while the unique cardinality is $||M_1|| = |Supp(M)| = 4$.

2.2 Languages and Event Logs

In this section, we introduce the fundamental concepts used to describe and reason about process behaviour in terms of sequences of actions. We begin by defining alphabets and traces, then formalise the notion of language as a multiset of traces, and extend it to the probabilistic setting through stochastic languages. The notation used in this section is based on [74] for the non-stochastic definitions, and on [52] for the stochastic extensions.

Definition 2.1 (Alphabet) An alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is defined as a finite non-empty set of symbols.

Definition 2.2 (Trace) A trace $\sigma \in \Sigma^*$ is a non-empty finite sequence over an alphabet Σ.

Definition 2.3 (Trace concatenation) Let $\sigma_i = \langle a_1, \ldots, a_n \rangle$ and $\sigma_j = \langle b_1, \ldots, b_m \rangle$ be two sequences over a alphabet Σ. Their concatenation, denoted $\sigma_i.\sigma_j$, is defined as:

$$\sigma_i.\sigma_j = \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle.$$

Definition 2.4 (Trace interleaving) We denote the set of all possible interleaving of σ_i and σ_j, written $\sigma_1 \diamondsuit \sigma_2$, inductively as follows:

- Base cases:

$$\varepsilon \diamondsuit \sigma_j = \{\sigma_j\}, \qquad \sigma_i \diamondsuit \varepsilon = \{\sigma_i\}$$

- Inductive step:

$$\text{For } \sigma_i = \langle a \rangle.\sigma_i' \text{ and } \sigma_j = \langle b \rangle.\sigma_j',$$
$$\sigma_i \diamondsuit \sigma_j = \left\{ \langle a \rangle.\gamma \mid \gamma \in \sigma_i' \diamondsuit \sigma_j \right\} \cup \left\{ \langle b \rangle.\gamma \mid \gamma \in \sigma_i \diamondsuit \sigma_j' \right\}$$

Example 2.1 (Concatenation and interleaving of traces).

Let $\sigma_1 = \langle a, b, c \rangle$ and $\sigma_2 = \langle d, e \rangle$ be two traces over the set S_1 with $|\sigma_1| = 3$ and $|\sigma_2| = 2$. Their concatenations are given by:

$$\sigma_1.\sigma_2 = \langle a, b, c, d, e \rangle \qquad \sigma_2.\sigma_1 = \langle d, e, a, b, c \rangle$$

Their interleaving set is:

$$\begin{aligned} \sigma_1 \diamondsuit \sigma_2 = \{ &\langle a, b, c, d, e \rangle, \langle a, b, d, e, c \rangle, \langle a, d, e, b, c \rangle, \langle d, e, a, b, c \rangle, \langle a, b, d, c, e \rangle, \\ &\langle a, d, b, e, c \rangle, \langle d, a, e, b, c \rangle, \langle a, d, b, c, e \rangle, \langle d, a, b, e, c \rangle, \langle d, a, b, c, e \rangle \} \end{aligned}$$

Definition 2.5 (Language) A language $\mathcal{L} \in \mathbb{B}(\Sigma^*)$ is a non-empty multiset of traces over an alphabet Σ. The corresponding multiplicity of each trace is referred to as its frequency. The support of a language $\mathcal{L}$, denoted as $\mathcal{T}_\mathcal{L}$, is the set of unique traces in $\mathcal{L}$. Formally:

$$\mathcal{T}_\mathcal{L} := supp(\mathcal{L})$$

We define a language as a multiset of traces, rather than the classical definition of a language as a set of traces. This choice reflects that identical traces may occur multiple times, and their frequency carries meaningful information in the context of the thesis. Even in the case of a language where all traces are unique, which often occurs when dealing with specific structures such as process models, the language is still formally defined as a multiset, with each trace assigned a frequency of one.

Definition 2.6 (Strong equality of languages) Let $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{B}(\Sigma^*)$ be two languages. Strong equality holds when both the set of traces and their frequencies are equal:

$$\mathcal{L}_1 = \mathcal{L}_2 \quad \iff \quad \forall \sigma \in \Sigma^*,\ \mathcal{L}_1(\sigma) = \mathcal{L}_2(\sigma).$$

Definition 2.7 (Weak equality of languages) Let $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{B}(\Sigma^*)$ be two languages. Weak equality holds when the two languages contain the same set of unique traces, regardless of their frequencies:

$$\mathcal{L}_1 \equiv \mathcal{L}_2 \quad \iff \quad \mathcal{T}_{\mathcal{L}_1} = \mathcal{T}_{\mathcal{L}_2},$$

Definition 2.8 (Language concatenation) For two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we denote by $\mathcal{L}_1 \bigodot \mathcal{L}_2$ the language consisting of traces resulting from concatenating any unique trace of $\mathcal{L}_1$ with any unique trace of $\mathcal{L}_2$:

$$\forall \sigma_1 \in \mathcal{T}_{\mathcal{L}_1},\ \forall \sigma_2 \in \mathcal{T}_{\mathcal{L}_2} : \quad (\sigma_1.\sigma_2) \in (\mathcal{L}_1 \odot \mathcal{L}_2)$$

with frequency:

$$(\mathcal{L}_1 \odot \mathcal{L}_2)(\sigma_1.\sigma_2) = \mathcal{L}_1(\sigma_1) \cdot \mathcal{L}_2(\sigma_2)$$

Definition 2.9 (Language interleaving) As concatenation, also interleaving, $\diamondsuit$, is extended to language in the natural way. For two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we denote by $\mathcal{L}_1 \diamondsuit \mathcal{L}_2$ the language consisting of all traces obtained by interleaving any unique trace of $\mathcal{L}_1$ with any unique trace of $\mathcal{L}_2$:

$$\forall \sigma_1 \in \mathcal{T}_{\mathcal{L}_1},\ \forall \sigma_2 \in \mathcal{T}_{\mathcal{L}_2} : \quad (\sigma_1 \diamondsuit \sigma_2) \subset (\mathcal{L}_1 \diamondsuit \mathcal{L}_2)$$

with frequency:

$$\sigma \in \sigma_1 \diamondsuit \sigma_2, \qquad (\mathcal{L}_1 \diamondsuit \mathcal{L}_2)(\sigma) = \mathcal{L}_1(\sigma_1) \cdot \mathcal{L}_2(\sigma_2)$$

The product over frequencies is justified by the fact that these operators will only be used, in the remainder of this work, on languages where all trace frequencies are equal to one. The concatenation and interleaving operators are naturally extended to families of languages.

Definition 2.10 (Stochastic language) A stochastic language over Σ is a function $\mathcal{S} \colon \Sigma^* \to [0, 1]$ such that $\sum_{\sigma \in \Sigma^*} \mathcal{S}(\sigma) = 1$. In other words, $\mathcal{S}$ defines a discrete probability distribution over the set of all finite sequences over Σ. Each value $\mathcal{S}(\sigma)$ represents the probability of the trace σ.

Although a stochastic language is formally defined as a function, by abuse of notation, we may represent it using multiset-like notation, where the exponent associated with each trace denotes its probability rather than its frequency. Similarly, we extend the notion of the support of a language $\mathcal{T}$ to the stochastic case. For a stochastic language $\mathcal{S}$ over the alphabet Σ^*, we denote by $\mathcal{T}_{\mathcal{S}}$ the set of traces

$$\mathcal{T}_{\mathcal{S}} = \{\sigma \in \Sigma^* \mid \mathcal{S}(\sigma) > 0\}$$

For example, if we define S_1 over the alphabet $\Sigma = \{a, b, c\}$ such that:

$$S_1(\langle a, b \rangle) = 0.6; \qquad S_1(\langle b, c \rangle) = 0.4; \qquad \forall \sigma \in \Sigma^* \setminus \{\langle a, b \rangle, \langle b, c \rangle\},\ S_1(\sigma) = 0$$

then we represent $S_1 = [\langle a, b \rangle^{0.6}, \langle b, c \rangle^{0.4}]$, and its support is $\mathcal{T}_{\mathcal{S}_1} = \{\langle a, b \rangle, \langle b, c \rangle\}$.

It is possible to derive a stochastic language from a finite language. Given a language $\mathcal{L}$, i.e., a multiset of traces, we can define a stochastic language $\mathcal{S}_{\mathcal{L}} \colon \mathcal{T}_{\mathcal{L}} \to [0, 1]$ by normalising the frequencies of the traces:

$$\mathcal{S}_{\mathcal{L}}(\sigma) = \frac{\mathcal{L}(\sigma)}{|\mathcal{L}|}$$

This yields a discrete probability distribution over the observed traces in the language.

Event log as a mathematical abstraction. We bridge here the gap between the raw data representation of event logs and their mathematical abstraction. Formally, each case corresponds to a trace $\sigma \in \Sigma_L^*$, that is, a finite sequence of activities drawn from the log's activity alphabet Σ_L. The event log L can therefore be regarded as a multiset of traces, where the frequency of each trace equals the number of cases in the log that exhibit the corresponding sequence of activities. This induces two key abstractions:

1. the *language* $\mathcal{L}_L$, i.e., the set of all traces contained in L, and
2. the *stochastic language* $\mathcal{S}_L$, i.e., a probability distribution over $\mathcal{L}_L$ obtained by normalising trace frequencies.

Hence, the raw event log can be mapped to the formal object $(\mathcal{L}_L, \mathcal{S}_L)$, which jointly captures the control-flow perspective and the stochastic behaviour of the underlying process.

Example 2.2 (Log's stochastic language).

Consider a toy event log L_1, defined over the activity alphabet

$$\Sigma_{L_1} = \{a, b, c, d, e, f\}$$

and let us assume that the log consists of 5 unique traces over 100 cases, with frequencies as in the following log language:

$$\mathcal{L}_{L_1} = [\langle a, c, d, e\rangle^{30}, \langle b, c, d, e\rangle^{25}, \langle c, b, d, f\rangle^{20}, \langle a, c, c, d, e\rangle^{15}, \langle c, a, c, d, f\rangle^{10}] .$$

from which we obtain the corresponding stochastic language through normalisation of the trace frequencies:

$$\mathcal{S}_{L_1} = [\langle a, c, d, e\rangle^{0.3}, \langle b, c, d, e\rangle^{0.25}, \langle c, b, d, f\rangle^{0.2}, \langle a, c, c, d, e\rangle^{0.15}, \langle c, a, c, d, f\rangle^{0.1}] .$$

2.3 Control-flow Process Mining

This section introduces the formal structures used to represent process models, the techniques for evaluating their conformance with event logs, and some of the most prominent methods for process discovery.

2.3.1 Process Models

The definitions and notations used in this section are based on [59] for labelled Petri net, on [72] and [81] for workflow net properties, and on [49] for process trees.

2.3.1.1 Labelled Petri Nets

A labelled Petri net [59] is a directed bipartite graph composed of two types of nodes: places, represented as circles, and transitions, depicted as rectangles. The net structure is defined by input arcs, which connect places to transitions, and output arcs, which connect transitions to places. Places may contain tokens. The distribution of tokens across the places defines the marking of the net, and the evolution of these tokens through transitions models the system's dynamics.

Definition 2.11 (Labelled Petri Net) Formally, a labelled Petri net describes a discrete-state model by means of a tuple $N = (P, T, F, \Sigma_N, \lambda, m_0)$, where :

- P is a finite set of places,
- T is a finite set of transitions, with $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs,
- Σ_N is an alphabet of labels,
- $\lambda : T \to \Sigma_N \cup \{\tau\}$ is a labelling function that assigns to each transition either a visible label in Σ_N or the silent label τ,
- $m_0 \in \mathbb{B}(P)$ is a multiset of places.

In the context of this thesis, we restrict the arc function F to arcs with multiplicity equal to 1. Moreover, we require the labelling function λ to be injective on $\lambda^{-1}(\Sigma_N)$, i.e., no two distinct transitions share the same visible label. Graphically, a transition is depicted in white if it is labelled with an element from Σ_N, and in black if it is labelled with τ (i.e., a silent transition). In the remainder, we refer to non-silent transitions through their unique labels, since λ is injective on Σ_N.

Semantics. A state of a labelled Petri net model consists of the distribution of tokens, represented as black dots, over its places, and is given by a multiset of places m, called a marking, with $m(p)$ being the number of tokens in place $p \in P$ in marking m. The initial marking is m_0. A marking change corresponds to the firing of a transition. The preset of a transition $t \in T$ is the set of its input places, i.e.,

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

Respectively, the postset of a transition $t \in T$ is the set of its output places, i.e.,

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

A transition $t \in T$ is enabled (and may be fired) in marking m if all of its input places contain at least one token, formally, if $\forall p \in {}^\bullet t, m(p) > 0$. Accordingly, $en(m) = \{t \in T \mid \forall p \in {}^\bullet t, m(p) > 0\}$ is the set of transitions that are enabled in m. Firing of an enabled transition t in marking m yields a new marking m', written as $m[t\rangle m'$, where m' results from m by removing a token from each input place and adding a token to each output place. Formally,

$$m'(p) = \begin{cases} m(p) - 1 & \forall p \in {}^\bullet t \cap (T \setminus t^\bullet), \\ m(p) + 1 & \forall p \in t^\bullet \cap (T \setminus {}^\bullet t), \\ m(p) & \forall p \in ({}^\bullet t \cap t^\bullet) \text{ or } p \notin ({}^\bullet t \cup t^\bullet) \end{cases}$$

Reachability. A marking m' is said to be reachable from a marking m if and only if there exists a firing sequence $\langle t_0, \ldots, t_n \rangle \in T^*$ such that there are markings $m_0, \ldots, m_{n+1}$ with

$$m_0 = m, \quad m_{n+1} = m', \quad \text{and} \quad m_i[t_i\rangle m_{i+1} \ \ \forall i = 0, \ldots, n.$$

For a Petri net N, we denote by $R_s(N)$ the reachability set of N, that is the set of markings m' reachable from the initial one m_0 and $R_g(N) = (R_s(N), A)$ the reachability graph of N where $A \subseteq R_S(N) \times R_S(N) \times T$ is the set of arcs whose elements $(m, m', t) \in A$ are such that $m[t\rangle m'$.

Safeness. A Petri net N is said to be safe if, for every reachable marking $m \in R_s(N)$, and for every place $p \in P$, we have $m(p) \leq 1$. The definition of safeness can be extended to k-safeness: a Petri net is k-safe if, for every reachable marking $m \in R_s(N)$ and every place $p \in P$, we have $m(p) \leq k$.

Example 2.3 (Petri net).

Figure 2.1 shows a labelled Petri net $N_1 = (P, T, F, \Sigma_{N_1}, \lambda, m_0)$ that describes the use of a common resource (modelised by place p_3) between two competitive processes (modelised by places p_1 and p_2) where:

$$P = \{p_1, p_2, p_3, p_4, p_5\}, \qquad T = \{t_1, t_2, t_3, t_4\},$$
$$F = \{(p_1, t_1), (p_2, t_2), \ldots, (t_3, p_1), (t_4, p_2)\}, \quad \Sigma_{N_1} = \{s_1, s_2, e_1, e_2\},$$
$$\lambda = \{t_1 \mapsto s_1,\ t_2 \mapsto s_2,\ t_3 \mapsto e_1,\ t_4 \mapsto e_2\}, \quad m_0 = [p_1, p_2, p_3].$$

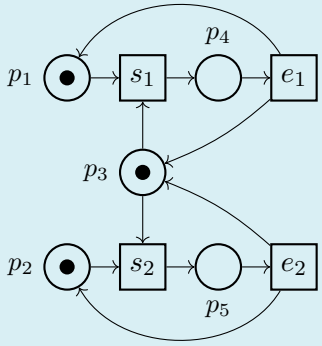


Figure 2.1 – Petri net N_1

Figure 2.2 and 2.3 respectively give the reachability set $R_s(N_1)$ and the reachability graph $R_g(N_1)$ of N_1.

$$m_0 = [\ p_1, p_2, p_3\ ]$$
$$m_1 = [\ p_2, p_4\ ]$$
$$m_2 = [\ p_1, p_5\ ]$$

Figure 2.2 – Reachability set of N_1

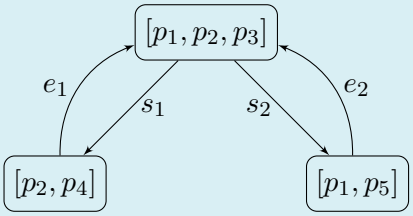


Figure 2.3 – Reachability graph of N_1

The preset of t_1 is $\{p_1, p_3\}$, and its postset is $\{p_4\}$. This means that t_1 is enabled when there is a token in both p_1 and p_3. Firing t_1 consumes these two tokens

and produces a new one in p_4. An example of a firing sequence is $\langle t_1, t_3 \rangle$, which describes the complete execution of one of the processes: the token in p_1 represents the start of the process, it consumes a shared resource from p_3, proceeds through execution, and eventually returns the resource, bringing the net back to its initial marking m_0. The net is safe: in all three reachable markings, no place contains more than one token.

2.3.1.2 Workflow nets

Since we are interested in modelling finite executions of a process, we rely on workflow nets [72, 81] (WN), a subclass of structured labelled Petri nets specifically designed to model processes with a clear notion of start and end. They ensure that every execution of the net moves the single token of the initial marking from the *source* place to the unique final *sink* place.

Definition 2.12 (Workflow net) A WN is a safe labelled Petri net $N = (P, T, F, \Sigma_N, \lambda, m_0)$, with the following constraints:

1. there exists a unique place, denoted $source$ with no incoming transitions (i.e., $\bullet source = \emptyset$) and a unique place denoted $sink$ with no outgoing transitions (i.e., $sink\bullet = \emptyset$),
2. the initial marking is $m_0 = [source]$ and,
3. every node $n \in (P \cup T)$ lies on an execution path from $[source]$ to $[sink]$.

Those constraints alone do not prevent behavioural anomalies such as deadlocks (no transition is enabled in a marking that is not the final marking), dead transitions (which can never fire in any reachable marking of the net), or tokens left behind (meaning that the sink place may contain a token while tokens remain in another place). To ensure meaningful execution semantics, we refer to the notion of soundness for a workflow net as follows:

Definition 2.13 (Sound Workflow Net) A workflow net is said to be sound if it satisfies the following three conditions:

1. Option to complete: From the initial marking m_0, it is possible to reach the final marking $m_f = [sink]$.
2. Proper completion: If a reachable marking m such as $m(sink) > 0$ is reached then $m = [sink]$.
3. No dead transitions: Every transition $t \in T$ occurs in at least one firing sequence from the initial marking $m_0 = [source]$ to the final marking $m_f = [sink]$.

In the remainder of this thesis, we shall always rely on contexts that guarantee the soundness of workflow nets, either by construction or through the enforcement of additional properties. In cases where soundness is not ensured, this will be explicitly stated.

Workflow net language. Every firing sequence $\langle t_1, t_2, \ldots, t_n \rangle \in T^*$ that leads from the initial marking $m_0 = [source]$ to the final marking $m_f = [sink]$ induces a trace

$$\sigma = \langle \lambda(t_1), \lambda(t_2), \ldots, \lambda(t_n) \rangle_{\setminus \tau}.$$

Here, the notation $\langle \cdot \rangle_{\setminus \tau}$ denotes the projection of the sequence of labels onto the visible alphabet Σ_N, i.e., all occurrences of the silent label τ are removed and do not appear in the resulting trace. The *language* of a workflow net N is then defined as the set of all such traces:

$$\mathcal{L}_N = \left\{ \langle \lambda(t_1), \ldots, \lambda(t_n) \rangle_{\setminus \tau} \;\middle|\; \begin{array}{l} \exists\, m_1, \ldots, m_n \in R_S(N) \\ [source][t_1\rangle m_1,\ m_n[t_n\rangle[sink] \\ m_i[t_{i+1}\rangle m_{i+1} \quad \forall i = 1, \ldots, n-2 \end{array} \right\}.$$

It is important to note that the language of a workflow net can be infinite, depending on the net's structure. In particular, loops (i.e., transitions and places forming cyclic execution paths) allow the generation of arbitrarily long traces by repeating certain portions of behaviour. As a result, the workflow net language may contain an unbounded number of distinct traces, even though each trace is finite.

Example 2.4 (Workflow net).

Figure 2.4 shows a workflow net $N_2 = (P, T, F, \Sigma_{N_2}, \lambda, m_0)$ consisting of 8 places, 6 labelled transitions and 3 silent transitions.

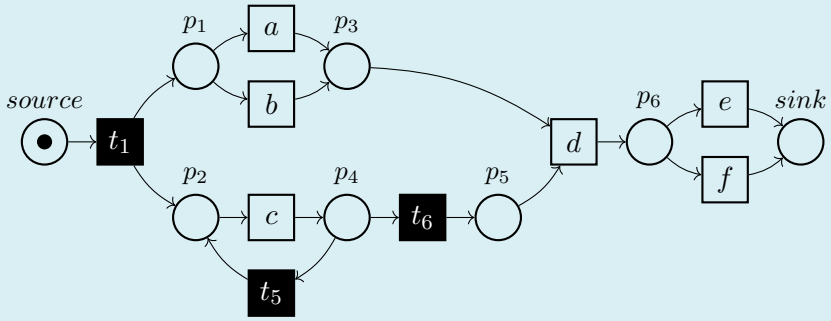


Figure 2.4 – Workflow net N_2

The reachability set in Figure 2.5 and the reachability graph in Figure 2.6 allow us to verify the constraints and the soundness of N_2. The initial marking $[source]$ is correctly represented with no incoming edges in the reachability graph, while the unique final marking $[sink]$ is terminal, meaning that no transitions are enabled once reached. Moreover, we observe that each transition appears on at least one execution path from $[source]$ to $[sink]$, ensuring that no transition is dead.

Note that Σ_{N_2} is equal to Σ_{L_1} (from Example 2.2) and that $\mathcal{T}_{\mathcal{L}_1} \subset \mathcal{T}_{\mathcal{L}_{N_2}}$, meaning that the workflow net can reproduce all the traces contained in the language $\mathcal{L}_1$. Moreover, the language of N_2 is infinite, due to the presence of the loop pattern that allows transition c to be repeated an arbitrary number of times.

Finally, notice that distinct firing sequences generate certain traces. For example, trace $\langle c, a, c, d, f \rangle$ is generated by both the following sequences of the net's transitions:

$$\langle t_1, c, t_5, a, c, t_6, d, f \rangle \qquad \langle t_1, c, a, t_5, c, t_6, d, f \rangle$$

$$\begin{aligned}
m_0 &= [\ source\ ] \\
m_1 &= [\ p_1, p_2\ ] \\
m_2 &= [\ p_2, p_3\ ] \\
m_3 &= [\ p_1, p_4\ ] \\
m_4 &= [\ p_3, p_4\ ] \\
m_5 &= [\ p_1, p_5\ ] \\
m_6 &= [\ p_3, p_5\ ] \\
m_7 &= [\ p_6\ ] \\
m_f &= [\ sink\ ]
\end{aligned}$$

Figure 2.5 – Reachability set of N_2

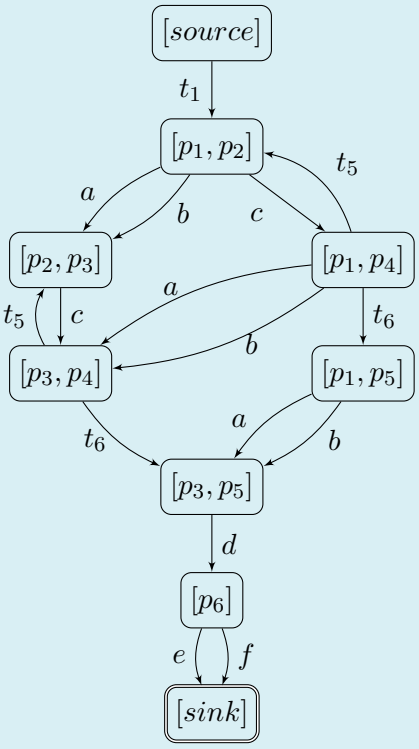


Figure 2.6 – Reachability graph of N_2

2.3.1.3 Process trees

Process trees [49] (PT) are a formalism used to obtain a hierarchical representation of a language built on top of an alphabet. The leaves of a PT correspond to the alphabet element, while the internal nodes correspond to operators through which the languages of the corresponding sub-trees are combined. Four operators are commonly used to construct process trees, namely *sequence* ($\rightarrow$), *choice* ($\times$), *parallel* ($\wedge$), and *loop* ($\circlearrowleft$).

Definition 2.14 (Process tree) Let Σ_Q be an alphabet of labels and $\{\rightarrow, \times, \wedge, \circlearrowleft\}$ the set of process tree operators. The set of process trees over Σ_Q is recursively defined as:

- if $a \in \Sigma_Q \cup \{\tau\}$ then $Q = a$ is a process tree,
- if $Q_1, \ldots, Q_n$ with $n \geq 2$ are process trees and $\oplus \in \{\rightarrow, \times, \wedge\}$ then $\oplus(Q_1, Q_2, \ldots, Q_n)$ is a process tree,
- if Q_1 and Q_2 are process trees then $\circlearrowleft (Q_1, Q_2)$ is a process tree.

Semantics of process trees. Let Q be a process tree over the alphabet Σ_Q. The language of Q, denoted by $\mathcal{L}_Q$, is defined recursively as:

- if $Q = a$ with $a \in \Sigma_Q$ then $\mathcal{L}_Q = \{\langle a \rangle\}$,
- if $Q = \tau$ then $\mathcal{L}_Q = \{\varepsilon\}$,
- (sequence:) if $Q = \rightarrow (Q_1, Q_2, \ldots, Q_n)$ then

$$\mathcal{L}_Q = \bigodot_{i=1}^{n} \mathcal{L}_{Q_i}$$

- (choice:) if $Q = \times(Q_1, Q_2, \ldots, Q_n)$ then

$$\mathcal{L}_Q = \bigcup_{i=1}^{n} \mathcal{L}_{Q_i}$$

- (parallel:) if $Q = \wedge(Q_1, Q_2, \ldots, Q_n)$ then

$$\mathcal{L}_Q = \diamondsuit_{i=1}^{n} \mathcal{L}_{Q_i}$$

- (loop:) if $Q = \circlearrowleft (Q_1, Q_2)$ then

$$\mathcal{L}_Q = \{\sigma_1.\sigma'_1.\sigma_2.\sigma'_2. \ \ldots \ .\sigma'_{m-1}.\sigma_m \mid m \geq 1, \forall i, 1 \leq i \leq m, \sigma_i \in \mathcal{L}_{Q_1}, \forall j, 1 \leq j \leq m-1, \sigma'_j \in \mathcal{L}_{Q_2}\}$$

Note that, differently from the original definition of PTs [49], in the context of this thesis, we opted for a binary ($\circlearrowleft (Q_1, Q_2)$), rather than an n-ary ($\circlearrowleft (Q_1, Q_2, \ldots, Q_n)$) version for the loop operator. In the n-ary version, the first child Q_1 represents the body of the loop (i.e., the language to be repeated), while the remaining $n - 1$ children $Q_2, \ldots, Q_n$ specify the loop conditions that enable each repetition. We point out that this does not affect the expressiveness of the PT formalism as any n-arguments loop PT can be expressed through a combination of binary loop PT where the first argument amongst the n considered ones is "looped" with a choice operator applied to the remaining $n - 1$ arguments, that is:

$$\circlearrowleft (Q_1, Q_2, \ldots, Q_n) = \circlearrowleft (Q_1, \times(Q_2, \ldots, Q_n))$$

Example 2.5 (Process tree).

Figure 2.7 describes a process tree Q_1 over the alphabet $\Sigma_{Q_1} = \{a, b, c, d, e, f\}$, formally defined as:

$$Q_1 = \rightarrow (\wedge(\times(a, b), \circlearrowleft (c, \tau)), d, \times(e, f))$$

Q_1 contains six labelled leaves and one silent leaf (τ). It models two choices, one loop, one parallel block, and a top-level sequence.

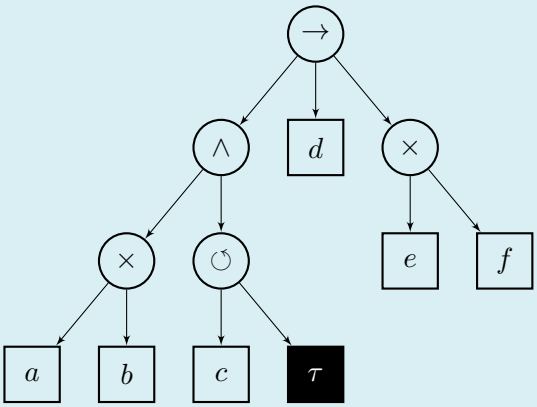


Figure 2.7 – Process tree Q_1

- Leaf nodes define base languages. A labelled leaf returns the singleton language containing only the trace with the corresponding activity. A silent leaf (τ) corresponds to the singleton language containing the empty trace ε.

$$Q_a = (a) \text{ with } \mathcal{L}_{Q_a} = [\langle a \rangle]$$
$$Q_b = (b) \text{ with } \mathcal{L}_{Q_b} = [\langle b \rangle]$$
$$Q_\tau = (\tau) \text{ with } \mathcal{L}_{Q_\tau} = [\varepsilon]$$

- Choice operators ($\times$) define the union of the languages of their children. A trace in the resulting language corresponds to a trace from exactly one of the branches.

$$Q_{c_1} = (\times(a, b)) \text{ with } \mathcal{L}_{Q_{c_1}} = [\langle a \rangle, \langle b \rangle]$$
$$Q_{c_2} = (\times(e, f)) \text{ with } \mathcal{L}_{Q_{c_1}} = [\langle e \rangle, \langle f \rangle]$$

- Loop operators ($\circlearrowleft$) define an infinite language where a trace consists of a sequence of one or more executions of the loop body (left child), and each repetition except the last must be followed by a trace from the loop condition (right child). The repetition can occur zero or more times. This structure ensures that the body is executed at least once, and may be repeated further, but only if each repetition is triggered by an execution of the loop condition.

$$Q_{l_1} = (\circlearrowleft (c, \tau)) \text{ with } \mathcal{L}_{Q_{l_1}} = [\langle c \rangle, \langle c, c \rangle, \langle c, c, c \rangle, \ldots]$$

- Parallel operators ($\wedge$) generate the set of all possible interleavings of one trace from each child language, preserving the relative order of labels within each branch.

$$Q_{p_1} = (\wedge(\times(a, b), \circlearrowleft (c, \tau)))$$
$$\text{with } \mathcal{L}_{Q_{p_1}} = [\langle a, c \rangle, \langle c, a \rangle, \langle b, c \rangle, \langle c, b \rangle, \langle a, c, c \rangle, \langle c, a, c \rangle, \langle c, c, a \rangle, \ldots]$$

- Sequence operators ($\rightarrow$) produce traces by concatenating one trace from each child language, in left-to-right order.

$$\mathcal{L}_{Q_1} = [\langle a, c, d, e\rangle, \langle b, c, d, e\rangle, \langle c, b, d, f\rangle, \\ \langle a, c, c, d, e\rangle, \langle c, a, c, d, f\rangle, \ldots]$$

Note that Q_1 is behaviourally equivalent to the workflow net N_2, as both induce the same language: $\mathcal{L}_{Q_1} = \mathcal{L}_{N_2}$ with identical alphabets $\Sigma_{Q_1} = \Sigma_{N_2}$. This means that Q_1 is capable of generating precisely the same set of traces as N_2, and in particular, it can reproduce all the traces contained in the language $\mathcal{L}_1$.

2.3.2 Control-flow Conformance Checking Metrics

Now that we have defined the models and their language semantics, we can assess their conformance with an event log using several metrics. Fitness and precision [18, 24] are two key metrics used to quantify the conformance of a process model concerning an event log, based on behavioural characteristics. Let us consider the language of an event log L, denoted by $\mathcal{L}_L$, and the language of a process model N, denoted by $\mathcal{L}_N$, both defined over an alphabet of activities Σ. The support (i.e., the set of unique traces of each language) is denoted respectively by $\mathcal{T}_{\mathcal{L}_L} \subseteq \Sigma^*$ and $\mathcal{T}_{\mathcal{L}_N} \subseteq \Sigma^*$.

Fitness measures the model's ability to reproduce the observed traces in the log. A language-based definition of fitness is given by:

$$Fitness(\mathcal{L}_L, \mathcal{L}_N) = \frac{|\mathcal{T}_{\mathcal{L}_L} \cap \mathcal{T}_{\mathcal{L}_N}|}{|\mathcal{T}_{\mathcal{L}_L}|}$$

This corresponds to the proportion of log traces that the model accepts. A fitness value of 1 indicates that the model can replay all log traces, meaning that $\mathcal{T}_{\mathcal{L}_N} \subset \mathcal{T}_{\mathcal{L}_L}$. A fitness value of 0 indicates that the model is unable to reproduce any of the log traces, meaning that $\mathcal{T}_{\mathcal{L}_L} \cap \mathcal{T}_{\mathcal{L}_N} = \emptyset$.

Precision evaluates how much additional behaviour the model allows beyond what is seen in the log. It is defined as:

$$Precision(\mathcal{L}_L, \mathcal{L}_N) = \frac{|\mathcal{T}_{\mathcal{L}_L} \cap \mathcal{T}_{\mathcal{L}_N}|}{|\mathcal{T}_{\mathcal{L}_N}|}$$

A precision score of 1 means that the log supports every trace the model generates. Lower values indicate that the model overgeneralises, allowing traces that were not observed.

In practice, computing these quantities exactly is often infeasible due to the potentially infinite nature of both languages. Approximate methods, such as alignment-based conformance checking or token-based replay, are typically employed to estimate these scores.

In addition to behavioural conformance, two other important dimensions are simplicity [57] and generalisation [66]. Simplicity refers to the structural complexity of the discovered model. Overly complex models are more challenging to interpret and analyse, particularly in practical settings. Generalisation, on the other hand, evaluates the model's ability to represent the observed behaviour and reasonable, unseen behaviour that is likely to occur in the underlying process. A model that overfits the log may score high on fitness but low on generalisation, failing to capture the actual process dynamics.

2.3.3 Process Discovery Methods

Now that we have introduced several ways of measuring the conformance of a model, we turn to algorithms for process discovery. These algorithms rely on different paradigms and strategies, but share the common goal of discovering a model that achieves high conformance with respect to the event log.

2.3.3.1 Relation-based algorithms

Alpha Miner. One of the earliest and most influential process discovery algorithms is the Alpha Miner [71]. The algorithm models the direct ordering relations between activities observed in the event log and can detect sequential, conflicting, and concurrent behaviour patterns by analysing directly-follows dependencies across the entire log. These relations are determined through a complete exploration of all traces in the event log. The Alpha Miner defines four main types of relations between activities:

- **Directly-follows relation** ($a > b$) when activity a is immediately followed by activity b in at least one trace.
- **Causality relation** ($a >> b$) inferred when a directly precedes b ($a > b$), but not the other way around ($b \not> a$). This typically represents a sequential dependency.
- **Concurrency relation** ($a \parallel b$) inferred when both $a > b$ and $b > a$ are observed, indicating that a and b can occur in any order, i.e., they are concurrent.
- **No-dependency relation** ($a \# b$): neither $a > b$ nor $b < a$ occurs, indicating that a and b are mutually exclusive and may represent a choice point in the process.

The algorithm builds separate sets for each type of relation and applies a series of simple rules to generate the corresponding workflow net. The construction does not involve silent transitions, which limits the expressiveness of the resulting model and prevents the correct representation of certain routing constructs, such as loops or skipped activities. In fact, a strict application of the algorithm to real-life logs usually yields unsound workflow nets in which the final marking is not reachable and in which several transitions can never fire.

Heuristic Miner. To overcome the limitations of the Alpha Miner, particularly its sensitivity to noise and its inability to handle real-life non-ideal logs, the Heuristic Miner [86] was introduced. Unlike the Alpha Miner, which relies solely on the existence of directly-follows relations, the Heuristic Miner incorporates frequency-based heuristics to distinguish between dominant and exceptional behaviour. This makes it more robust to noise and better suited for practical applications. The discovery process relies on constructing a dependency graph, in which edges represent dependency measures between pairs of activities. The strength of a dependency from activity a to b is typically computed using the formula:

$$\text{dep}(a, b) = \frac{|\{a > b\}| - |\{b > a\}|}{|\{a > b\}| + |\{b > a\}| + 1}$$

where $\{a > b\}$ denotes the number of times activity a is directly followed by activity b in the log. This score reflects the likelihood that a causally precedes b. By introducing thresholds over the dependency values, the Heuristic Miner filters out low-frequency relations and focuses on the most representative paths in the log. This makes it more robust to noise and outliers, at the cost of potentially missing infrequent but relevant behaviours. The resulting model is typically expressed as a labelled Petri net or a causal net, and may include loops and other structures that the Alpha Miner cannot represent.

Despite its empirical strengths, the Heuristic Miner, similarly to the Alpha Miner, lacks formal guarantees of correctness or soundness and may produce models that contain deadlocks, livelocks, or other structural anomalies. Nevertheless, it remains one of the most widely used discovery techniques in practical settings due to its scalability on real-life event logs.

2.3.3.2 Block-structural-based algorithms

Inductive Miner. The Inductive Miner [49] represents a significant advancement in process discovery by enforcing a block-structured model semantics. A block structure model is defined as a hierarchical model that can be divided recursively into parts having single entry and exit points [49]. Unlike Alpha and Heuristic Miners, which may produce unstructured or unsound models, the Inductive Miner guarantees soundness and behavioural completeness by construction. In fact, by construction, the algorithm ensures that the discovered model has perfect fitness with respect to the log.

The algorithm recursively partitions the event log into subsets of traces that correspond to distinct control-flow constructs (e.g., sequence, choice, concurrency and loops), and builds a process tree that captures this hierarchical structure. At each recursion level, the algorithm identifies a cut in the log, which is a decomposition pattern that matches one of the predefined operators. This is achieved by first constructing the directly-follows graph of the log L, denoted by:

$$G(L) = (\Sigma_L, E)$$

where Σ_L is the set of log activities and $E \subseteq \Sigma_L \times \Sigma_L$ is the set of directly-follows relations observed in the log. An edge $(a, b) \in E$ is included if there exists at least

one trace in L where activity a is immediately followed by activity b ($a > b$). The search for cuts is then applied to the directly-follows graph, splitting it into smaller subgraphs by identifying cuts based on the form of the graph, that is, on recognisable structural patterns that reflect control-flow constructs. The recursion stops when the remaining subgraph contains only a single activity node. The collection of identified cuts is then used to connect all activities into a sound and block-structured process tree, which can subsequently be translated into a labelled Petri net. Thanks to its formal guarantees and consistent model quality, the Inductive Miner has become the default algorithm in many process mining tools and is widely used in both research and industry.

Control-flow patterns detected by Inductive Miner. The Inductive Miner detects four fundamental process tree operators, each corresponding to a control-flow construct that can be identified from the directly-follows graph of the log. These cuts are mutually exclusive and exhaustive, ensuring that every sublog can be recursively decomposed into smaller sublogs. For each cut, the corresponding language, process tree, and workflow net are shown in Figure 2.8.

- **Sequence cut ($\rightarrow$)**: activities are arranged in a strict order, such that all occurrences of activities in one partition always precede the activities in the next. The resulting process tree operator is a sequence node, and the corresponding Petri net fragment connects the transitions in a sequential chain.
- **Choice cut ($\times$)**: the log reveals that traces follow disjoint alternatives, i.e., exactly one among several groups of activities is executed. This is modelled as an exclusive choice operator in the process tree, and in the Petri net as a conflict structure where a single token enables only one outgoing transition.
- **Parallel cut ($\wedge$)**: the activities in the sublog can occur in any order, potentially interleaved, without enforcing a specific sequence. This is recognised when the directly-follows graph indicates mutual concurrency. It is modelled as a parallel (AND-split/AND-join) operator in the process tree, and as a Petri net fragment where multiple branches can fire independently.
- **Loop cut ($\circlearrowleft$)**: the log contains repetitions of a block of activities, separated by occurrences of a distinct "redo" activity or set of activities. This is detected by identifying cyclic patterns in the directly-follows graph. In the process tree, this yields a loop node with a body and a redo branch, while in the Petri net, the corresponding structure allows tokens to circulate back until the exit is taken.

Together, these four patterns are sufficient to decompose any log into a block-structured process tree recursively. Combined with the guarantees of the algorithm, this ensures that the discovered models are sound, well-structured, and behaviourally complete.

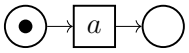
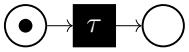
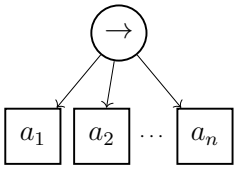
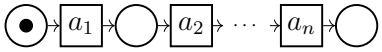
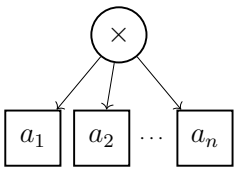
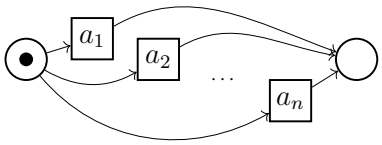
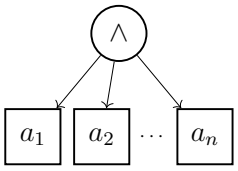
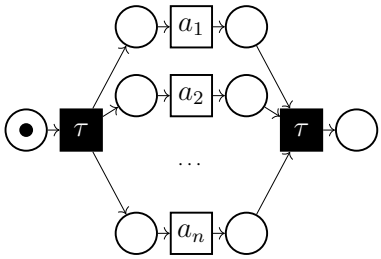
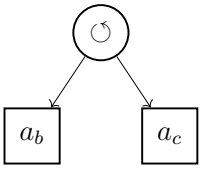
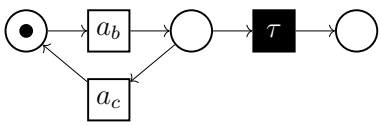
Language	Process tree	Workflow net
Activity		
$\mathcal{L} = [\langle a \rangle]$		
Silent activity		
$\mathcal{L} = [\varepsilon]$		
Sequence		
$\mathcal{L} = \bigodot_{i=1}^{n} [\langle a_i \rangle]$ $= [\langle a_1, a_2, \ldots, a_n \rangle]$		
Choice		
$\mathcal{L} = \bigcup_{i=1}^{n} [\langle a_i \rangle]$ $= [\langle a_1 \rangle, \langle a_2 \rangle, \ldots, \langle a_n \rangle]$		
Parallel		
$\mathcal{L} = \diamondsuit_{i=1}^{n} [\langle a_i \rangle]$ $= [\,\langle a_1, a_2, \ldots, a_n \rangle,$ $\ldots,$ $\langle a_n, \ldots, a_2, a_1 \rangle\,]$		
Loop		
$\mathcal{L} = [\langle a_b \rangle,$ $\langle a_b, a_c, a_b \rangle,$ $\langle a_b, a_c, a_b, a_c, a_b \rangle,$ $\ldots]$		

Figure 2.8 – Mapping of language patterns to their corresponding representations in process trees and workflow nets

Inductive Miner infrequent. One of the key advantages of the Inductive Miner is its robustness to noise, particularly when using its infrequent variant (IMf) [50], which filters out low-frequency behaviour that may otherwise disrupt the structural decomposition. The infrequent variant introduces an additional parameter: a threshold value $\kappa \in [0, 1]$ that is used to filter out infrequent relations. Based on the directly-follows graph $G(L)$ of the log, each arc (a, b) is evaluated by counting the number of times the relation $a > b$ occurs in the log. The most frequent relation in the graph is identified, and its frequency is multiplied by the threshold κ to determine a cutoff value. Any arc whose frequency falls below this cutoff is considered infrequent and is removed from the graph. Formally, we can build a new infrequent directly-follow graph, denoted by:

$$G_i(L, \kappa) = (\Sigma_L, E_\kappa), \quad \text{where} \quad E_\kappa = \left\{ (a, b) \in E \;\middle|\; |\{a > b\}| \geq \kappa \cdot \max_{(x,y) \in E} |\{x > y\}| \right\}.$$

After this filtering step, the search for cuts and the construction of the process tree proceed as usual. By removing low-frequency relations, the algorithm avoids overfitting and improves the generalisation of the discovered model. However, this also modifies the resulting language of the model, as traces containing infrequent relations may be partially excluded. Specifically, only the segments of traces involving the removed relations are affected, which can result in simplified but less behaviourally complete models. In particular, a substantial amount of causality information may be lost in the filtering process. This is because certain frequent relations can give rise to structural patterns that include infrequent transitions as part of their execution. As a result, removing infrequent relations may inadvertently break valid control-flow dependencies and affect the completeness of the discovered model.

Directly-Follows Model Miner. In an effort to more directly exploit the structure of the directly-follows model, and to enable its transformation into a labelled Petri net, the Directly-Follows Model Miner [54] (DFMM) was introduced. This method leverages the raw directly-follows relations extracted from the event log to construct a directly-follows representation that preserves both behavioural and statistical properties of the log. The trace-based version of the DFMM combines two key ideas: directly follows model construction and iterative trace filtering. The discovery procedure operates as follows:

1. A directly-follows model is initially constructed from the log. For each trace, nodes and edges are added accordingly, and edge frequencies are incremented based on their occurrence in the log.
2. The least frequent edges in the DFM are identified, as they are likely to represent exceptional or noisy conduct.
3. All traces that make use of these low-frequency edges are removed from the event log.

Those steps are repeated iteratively until a user-defined threshold is reached, indicating the maximum proportion of traces that may be removed from the log. This

iterative pruning ensures that the final directly-follows model fits at least the specified proportion of the original traces (assuming only complete traces are considered). This trace-coverage guarantee is a distinguishing feature of this variant and is generally not offered by other discovery techniques, which may overfit or underfit the log without explicit control.

2.4 Stochastic Process Mining

Since the focus of this thesis lies in the stochastic aspects of process mining, we recall here some fundamental elements in this context. In Section 2.4.1, we introduce the class of models commonly used in stochastic process discovery, namely the stochastic extension of workflow nets. In Section 2.4.2, we overview the stochastic extension of conformance necessary to assess the resemblance between the log's and the model's stochastic languages and describe two stochastic conformance measures, namely the so-called *Earth Mover's* and the *Kullback-Leibler divergence*. Finally, in Section 2.4.3, we briefly review existing stochastic process discovery approaches.

2.4.1 Stochastic Workflow Nets

Definition 2.15 (Stochastic workflow nets) A stochastic workflow net (sWN) is a tuple $S = (P, T, F, W, \Sigma_S, \lambda, m_0)$ where:

- $(P, T, F, \Sigma_S, \lambda, m_0)$ is a workflow net,
- $W : T \to \mathbb{R}_{>0}$ is a function assigning a weight to transitions.

In practice, it can also be seen as a generalised stochastic Petri net (GSPN) [59] consisting uniquely of immediate transitions. Transitions of an sWN fire with a probability that is a function of their weights. Specifically,

$$\mathbb{P}(t|m) = \frac{W(t)}{\sum_{t' \in en(m)} W(t')}$$

denotes the probability that enabled transition t fires in marking m.

Stochastic language of an sWN. We denote by $\mathcal{S}_S$ the stochastic language associated with the sWN S.

- The probability of a trace σ in $\mathcal{S}_S$ can be computed by summing the probabilities of all those transition firing sequences that move the token initially present in $source$ to $sink$ and generate σ. In most cases, the same trace can be generated by more than one transition sequence because of the presence of silent transitions.
- The probability of a firing sequence is given by the product of the firing probabilities of the individual transitions that compose the sequence.

Formally, consider a firing sequence of n transitions $\langle t_0, t_1, \ldots, t_n \rangle$ leading through a corresponding sequence of $n+1$ reachable markings $m_0, m_1, \ldots, m_f$ where m_0 is the initial marking and m_f is the final marking. The probability of this firing sequence is defined as:

$$\mathbb{P}(\langle t_0, t_1, \ldots, t_n \rangle) = \prod_{i=0}^{n} \mathbb{P}(t_i | m_i)$$

The stochastic language $\mathcal{S}_S$ depends on the weights of the transitions. In a practical implementation, the weights can be represented as a vector $\overline{w} \in \mathbb{R}_{>0}^{|T|}$ ($|T|$ being the number of transitions of S). We will denote by $\mathcal{S}_S(\sigma, \overline{w})$ the probability of trace σ when the weight function W of S assigns weights to the transitions according to $\overline{w}$.

Example 2.6 (Stochastic workflow net).

Figure 2.9 depicts a stochastic workflow net S_1, built from the control-flow structure of N_2 and enriched with a weight assignment function W.

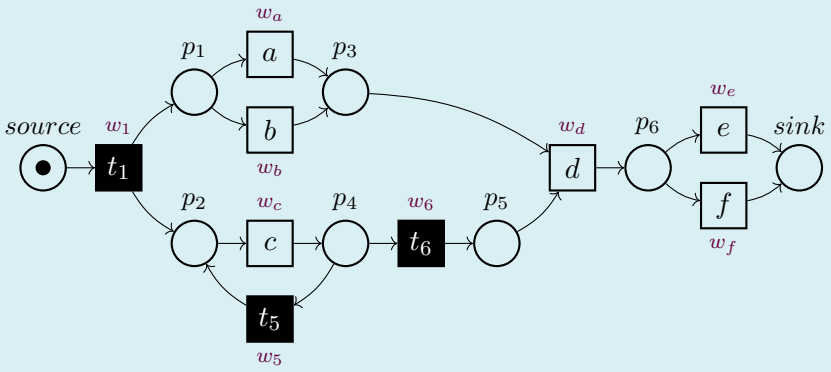


Figure 2.9 – Stochastic workflow net S_1

If we are interested in the probability of firing transition t_5 in marking $m_3 = [p_1, p_4]$, where transitions a, b, t_5, and t_6 are enabled, then:

$$\mathbb{P}(t_5 | m_3) = \frac{w_5}{w_a + w_b + w_5 + w_6}$$

Now, let us consider the probability of observing the trace $\sigma_1 = \langle c, a, c, d, f \rangle$ under the current weight assignment $\overline{w}$. The names of the markings are provided in the reachability set shown in Figure 2.5.

$$\begin{aligned}
\mathcal{S}_S(\sigma_1, \overline{w}) &= \mathbb{P}(\langle t_1, c, t_5, a, c, t_6, d, f \rangle) + \mathbb{P}(\langle t_1, c, a, t_5, c, t_6, d, f \rangle) \\
&= \mathbb{P}(t_1 | m_0) \cdot \mathbb{P}(c | m_1) \cdot \mathbb{P}(t_5 | m_3) \cdot \mathbb{P}(a | m_1) \\
&\quad \cdot \mathbb{P}(c | m_2) \cdot \mathbb{P}(t_6 | m_4) \cdot \mathbb{P}(d | m_6) \cdot \mathbb{P}(f | m_7) \\
&\quad + \mathbb{P}(t_1 | m_0) \cdot \mathbb{P}(c | m_1) \cdot \mathbb{P}(a | m_3) \cdot \mathbb{P}(t_5 | m_4) \\
&\quad \cdot \mathbb{P}(c | m_2) \cdot \mathbb{P}(t_6 | m_4) \cdot \mathbb{P}(d | m_6) \cdot \mathbb{P}(f | m_7) \\
&= \frac{w_1}{w_1} \cdot \frac{w_c}{w_a + w_b + w_c} \cdot \frac{w_5}{w_a + w_b + w_5 + w_6} \cdot \frac{w_a}{w_a + w_b + w_c} \\
&\quad \cdot \frac{w_c}{w_b + w_c} \cdot \frac{w_6}{w_5 + w_6} \cdot \frac{w_d}{w_d} \cdot \frac{w_f}{w_e + w_f}
\end{aligned}$$

$$+\frac{w_1}{w_1}\cdot\frac{w_c}{w_a+w_b+w_c}\cdot\frac{w_a}{w_a+w_b+w_5+w_6}\cdot\frac{w_5}{w_5+w_6}$$
$$\cdot\frac{w_c}{w_c}\cdot\frac{w_6}{w_5+w_6}\cdot\frac{w_d}{w_d}\cdot\frac{w_f}{w_e+w_f}$$

2.4.2 Stochastic Conformance

In the stochastic setting, conformance is assessed at the level of stochastic languages. The log induces an empirical probability distribution over traces, while the model defines a theoretical one over the same space. Stochastic conformance measures, therefore, compare not only which traces occur in both but also how closely the probabilities assigned by the model match those observed in the log. This extends classical notions of fitness and precision by incorporating the quantitative alignment of distributions.

Earth Mover's Stochastic Conformance. Within the process mining community, one of the most widely adopted probabilistic conformance measures is the Earth Mover's Stochastic Conformance (EMSC) metric [52, 53]. This measure is derived as an adaptation of the classical Earth Mover's Distance (EMD), also known as the first-order Wasserstein distance [62], to the setting of stochastic languages. Intuitively, the idea is to compute the minimal cost to transform one distribution into the other, where the cost is intended as the amount of probability mass that needs to be moved multiplied by the average distance it must be moved.

Example 2.7 (Earth Mover's Distance).

As an example, consider two discrete random variables, X and Y, both defined over the same support $\{1, 2, 3\}$, and associated with probability mass functions $p_X(x)$ and $p_Y(y)$, respectively, as illustrated in Figure 2.10.

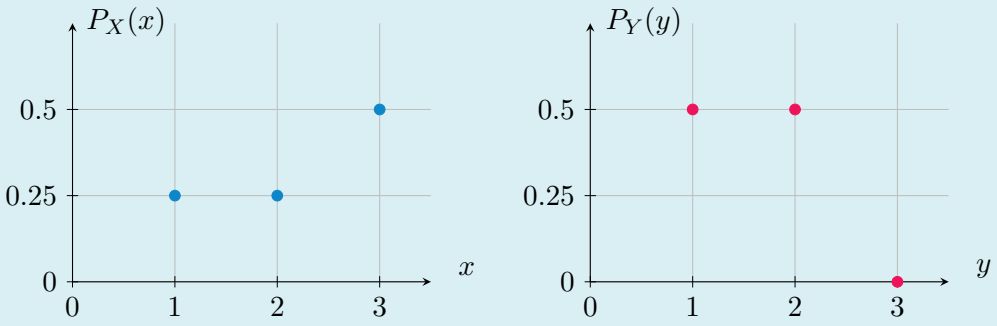


Figure 2.10 – Probability mass functions $P_X(x)$ and $P_Y(y)$

In this simple case, it is obvious that, in order to transform P_X to P_Y with minimal cost:

- 0.25 of probability mass needs to be moved from $x = 3$ to $x = 1$ with

associated distance $3 - 1 = 2$ and,

- 0.25 of probability mass needs to be moved from $x = 3$ to $x = 2$ with associated distance $3 - 2 = 1$.

The total cost is $0.25 \cdot 2 + 0.25 \cdot 1 = 0.75$ which is the EMD between P_X and P_Y. Transforming P_Y into P_X incurs the same minimal cost. This transformation can be represented by the so-called transport matrix M, which specifies how mass is transferred between the two distributions, and the associated cost matrix C, which defines the cost of transporting one unit of mass from one element to another. These matrices are, respectively, given by:

$$M = \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0.25 & 0.25 & 0 \end{pmatrix} \qquad C = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \tag{2.1}$$

Reading the transport matrix M row-wise describes how the probability mass from P_X is redistributed to form P_Y, while reading it column-wise reflects the transformation from P_Y back to P_X. By construction, the row sums of M correspond to the values of P_X and the column sums correspond to P_Y. The EMD between the two distributions can then be computed using the transport and cost matrices as follows:

$$EMD(P_X, P_Y) = \sum_{(i,j) \in \{1,2,3\}} C_{i,j} M_{i,j} = 3/4 \tag{2.2}$$

The Earth Mover's Stochastic Conformance (EMSC) measure compares the probability distributions induced by a log and a model, taking into account a ground distance that quantifies the dissimilarity between traces (e.g., edit distance or prefix alignment cost). The intuition is to evaluate the minimal cost of transforming the probability mass from the model distribution into that of the log, where the cost of moving mass is proportional to the behavioural distance between traces.

Formally, the first component of the measure is the reallocation function $r : \mathcal{S}_L \times \mathcal{S}_S \to [0, 1]$, which specifies how probability assigned to a model trace σ_S is transferred to a log trace σ_L, while preserving the total probability mass of both distributions. The second component is a trace distance function d that determines the cost of reallocating mass. It must be symmetric ($d(\sigma_S, \sigma_L) = d(\sigma_L, \sigma_S)$) and satisfy $d(\sigma, \sigma) = 0$. Its choice strongly influences the measure: a common simplification is the *unit distance*, where traces are either equal (0) or different (1), which reduces computational complexity but overlooks nuances of similarity. The resulting conformance metric is referred to as the unit Earth Mover's Stochastic Conformance measure (uEMSC). More generally, the most common distance is the Levenshtein distance [55], which counts the minimum number of edit operations needed to transform one trace into another. Since longer traces naturally allow for more edits, a normalised version is often used, obtained by dividing the raw distance by the length of the longer trace, yielding a value in $[0, 1]$ that reflects the proportion of differences:

$$d_{\text{norm\_lev}}(\sigma_1, \sigma_2) = \frac{d_{\text{lev}}(\sigma_1, \sigma_2)}{\max(|\sigma_1|, |\sigma_2|)}.$$

Example 2.8 (Levenshtein distance).

Consider $\sigma_1 = \langle a, b, c \rangle$ and $\sigma_2 = \langle a, a \rangle$ over $\Sigma = \{a, b, c\}$:

$$d_{\text{lev}}(\sigma_1, \sigma_2) = 2 \quad \text{(substitute } b \text{ with } a\text{, delete } c\text{)}.$$

Hence,

$$d_{\text{norm\_lev}}(\sigma_1, \sigma_2) = \tfrac{2}{3}.$$

The cost of a reallocation is

$$c(r, S, L) = \sum_{\sigma_S \in \mathcal{T}_S} \sum_{\sigma_L \in \mathcal{T}_L} r(\sigma_S, \sigma_L)\, d(\sigma_S, \sigma_L),$$

and EMSC selects the reallocation r minimising this cost among the infinite set of reallocation function $\mathcal{R}$:

$$EMSC(S, L) = 1 - \min_{r \in \mathcal{R}} c(r, S, L).$$

The search for the optimal reallocation function makes the computation of the EMSC equivalent to solving an optimisation problem, which can be computationally expensive. Since process models may generate infinitely many traces, a truncated variant (tEMSC) restricts the comparison to the subset of traces covering a user-defined portion of the probability mass (e.g., 0.8), ensuring that the measure remains computable.

Kullback–Leibler divergence. An information-theoretic way to quantify the discrepancy between a process model and an event log in the stochastic setting is to use divergence measures. Among these, the Kullback–Leibler divergence [42] (KLD) is widely employed to measure the difference between two probability distributions over the same domain.

Given two stochastic languages $\mathcal{S}_L$ and $\mathcal{S}_S$, representing respectively the empirical distribution derived from the log and the generative distribution induced by the model, the KLD from $\mathcal{S}_L$ to $\mathcal{S}_S$ is defined as:

$$D_{KL}(L \,\|\, S) = \sum_{\sigma \in \mathcal{T}_{\mathcal{S}_L}} \mathcal{S}_L(\sigma) \cdot \log \frac{\mathcal{S}_L(\sigma)}{\mathcal{S}_S(\sigma)}.$$

This value measures the information loss incurred when the model distribution $\mathcal{S}_S$ is used to approximate the empirical distribution $\mathcal{S}_L$. A value of zero indicates perfect agreement between the two distributions over the support of the log stochastic language $\mathcal{T}_{\mathcal{S}_L}$. However, KLD is asymmetric and becomes undefined whenever $\mathcal{S}_L(\sigma) > 0$ while $\mathcal{S}_S(\sigma) = 0$ for some trace σ, making it highly sensitive to missing support in the model. This issue can be alleviated by applying smoothing techniques or restricting the comparison to the common support of both distributions.

Despite these limitations, KLD remains an expressive and lightweight measure for comparing trace-level probability distributions. It has been used in prior work

both to guide model inference and to evaluate the statistical alignment between observed and simulated behaviour.

Entropy relevance. The entropic relevance [63, 46, 5] of a stochastic process model S with respect to an event log L measures the average number of bits required to encode a trace from L using the stochastic language induced by S. Formally,

$$ER(L \,\|\, S) \;=\; \sum_{\sigma \in \mathcal{T}_L} \mathcal{S}_L(\sigma) \cdot \mathsf{cost}(\sigma, S),$$

where

$$\mathsf{cost}(\sigma, S) \;=\; \begin{cases} -\log_2 \mathcal{S}_S(\sigma), & \text{if } \mathcal{S}_S(\sigma) > 0, \\ \mathsf{background\_cost}(\sigma), & \text{if } \mathcal{S}_S(\sigma) = 0. \end{cases}$$

The first case corresponds to the Shannon coding length of σ under the model S. The second case applies when σ cannot be generated by S, i.e., $\mathcal{S}_S(\sigma) = 0$. In that situation, the *background cost* provides a finite penalty obtained by resorting to a background distribution that guarantees finite values while punishing the model for missing traces present in the log.

2.4.3 Stochastic Discovery Methods

In the field of stochastic process discovery, the objective is to derive a stochastic process model from an event log that not only reproduces the control-flow relations between activities but also captures the frequency distribution of traces. To this end, various methods have been proposed, following either a direct or an indirect discovery strategy.

2.4.3.1 Direct stochastic process discovery

GDT-SPN miner. One of the earliest attempts to incorporate stochasticity from event logs into process models was the introduction of the Generally Distributed Transition Stochastic Petri Nets Miner [67] (GDT-SPN Miner). The core idea of this approach is to discover a generalised stochastic Petri net (GSPN) in which timed transitions can be associated with a wide range of probability distributions, such as uniform, normal, deterministic, or log-normal. The identification of these probability distributions and their parameters relies on the notion of alignments between log traces and model traces. Based on the observed values extracted from the event log, the algorithm infers the most likely stochastic laws and parameters that best explain the observations. A key strength of the method is its robustness against noise, as it was specifically designed to cope with imperfect event logs. This work in particular constitutes a core contribution to time modelling from event logs, as it was one of the first approaches to explicitly lift timestamp information from traces into parametric transition-time distributions within a stochastic process model.

Toothpaste Miner. A more recent and robust approach is proposed in [22]. The framework introduces a probabilistic extension of process trees, in which each node is annotated with parameters capturing the probabilistic relations observed in the event log. First, a preliminary translation of the log into a probabilistic process tree is performed. Then, a series of well-defined reduction and abstraction rules is iteratively applied to refine the tree. Finally, the resulting probabilistic process tree can be translated into a stochastic workflow net through a set of formal translation rules. These reduction and abstraction rules enable the method to handle complex structures, such as loops and concurrency patterns, while maintaining an overall model complexity at a manageable level.

2.4.3.2 Indirect stochastic process discovery

All the contributions presented in this thesis build upon indirect approaches. In recent years, several significant contributions have emerged in this line of research, proposing novel frameworks and techniques to enrich process models with probabilistic information.

Weight estimation. In their seminal work [21], Burke *et al.* proposed an indirect approach defining six different estimators, obtained by combining simple log-based statistics with structural properties of the discovered workflow net. This approach, termed weight estimation, decouples stochastic annotation from control-flow discovery and instantiates the framework with six different estimators that become increasingly sophisticated.

- The first estimator, frequency (w_{freq}), counts the number of times each activity occurs in the log.
- The second and third, the left-handed (w_{lhpair}) and right-handed (w_{rhpair}) activity-pair estimators, exploit frequencies of successor and predecessor activity pairs, thereby incorporating causal relations from the model's structure.
- The fourth, mean-scaled activity-pair ($w_{pairscale}$), normalises activity-pair frequencies by the average transition frequency in the log to improve comparability across logs of different sizes.
- The fifth, the fork distribution estimator (w_{fork}), distributes weights across competing branches based on the observed branching frequencies in the log.
- Finally, the alignment-based estimator (w_{align}) leverages alignments between traces and the model, counting how often transitions are visited in synchronous or model moves.

These estimators, while lightweight and computationally efficient, have been shown to generate stochastic models with conformance comparable to those of more complex approaches, making them a practical choice when scalability and speed are critical.

WaWE. A way more recent contribution to indirect stochastic process discovery is the Wasserstein Weight Estimation (WaWE) framework, introduced by Brockhoff *et al.* [17]. This approach optimises the weights of an sWN with respect to the Earth Mover's Stochastic Conformance (EMSC) measure. The key idea is to exploit the optimal transport theory underlying EMSC, which enables the use of subgradient-based optimisation methods. Specifically, the algorithm constructs a computational graph that links transition weights to trace probabilities, and then iteratively updates the weights by backpropagating the subgradients of a penalised EMSC loss (pEMSC). To address the challenges posed by loops and potentially infinite sets of traces, WaWE relies on sampling techniques to approximate the model's trace distribution, while introducing auxiliary traces to penalise residual probability mass and ensure stable optimisation. Experimental results demonstrate that WaWE is both computationally feasible and practical, achieving EMSC scores that often match or surpass those of state-of-the-art methods, such as GDT-SPN or Toothpaste Miner. In particular, WaWE demonstrates substantial improvements on highly variable logs, where event data contain a large number of distinct trace variants and diverse behavioural patterns between activities.

SLPN Miner. The work of Leemans *et al.* [45] further develops the optimisation perspective on stochastic process discovery. The authors provide a formal definition of the stochastic discovery problem, casting it as an optimisation task where the goal is to construct an sWN that maximises conformance with an input event log according to a chosen stochastic conformance measure. In particular, they distinguish two dimensions of optimisation: the discovery of the control-flow structure and the assignment of stochastic information (transition weights). This leads them to investigate two possible settings: the direct discovery procedure and the indirect one. For the direct setting, they reduce stochastic process discovery to a decision problem for the unit Earth Mover's Stochastic Conformance (uEMSC), formulating it as a non-convex optimisation over the structure and the weights of an sWN. Although this formulation guarantees models that maximise uEMSC, it is not computationally tractable in practice and remains mainly of theoretical interest. For the indirect setting, where a control-flow model is assumed as input, the authors propose a practically applicable optimisation technique. This method symbolically computes the probability of each log trace in terms of the sWN's weight parameters by constructing a cross-product between the model's stochastic reachability graph and a silenced deterministic finite automaton representing the trace. The resulting system of equations is then used to optimise stochastic conformance measures such as uEMSC or entropic relevance (ER-1), thereby yielding a stochastic net that best fits the observed log frequencies. Experimental results demonstrate that this optimisation-based formulation can outperform heuristic or estimator-driven approaches in terms of conformance of the discovered model. The main strength of this contribution lies in providing a formal notion of optimality in stochastic discovery, together with the first algorithms that achieve it in realistic indirect settings. A significant limitation, however, concerns scalability: the symbolic representation of log traces and the cross-product between their automata and the reachability graph

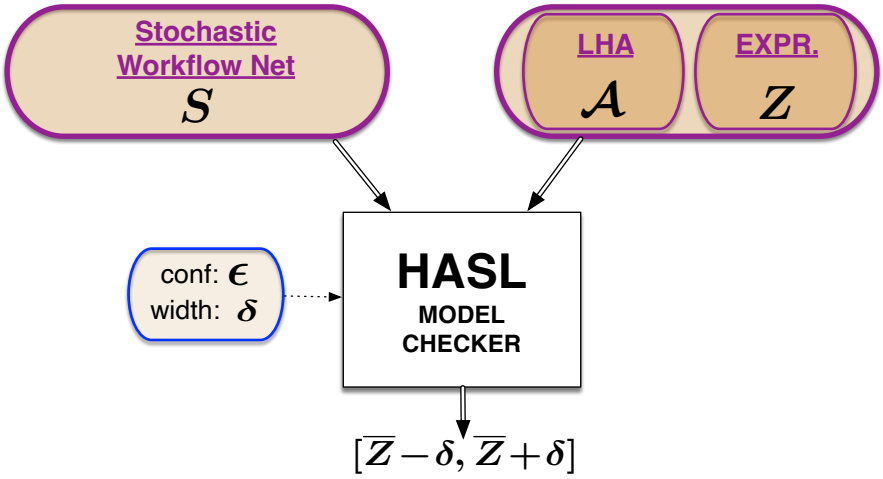


Figure 2.11 – The HASL statistical model checking scheme

quickly becomes intractable. In particular, when applied to real-life event logs, it is often infeasible to use this approach with perfectly fitting models, as the size of their reachability graphs, combined with the number of traces in the log, results in a cross-product that is too large to be handled in practice.

2.5 Techniques for Stochastic Analysis

One of the contributions presented in this thesis consists of a novel parameter inference framework for stochastic process discovery (Chapter 5). Since this framework is obtained via a combination of a statistical model checking approach based on the so-called *Hybrid Automata Stochastic Language* (HASL) with a Bayesian parameter inference method, namely the *Approximate Bayesian Computation* (ABC) method, we briefly outline both approaches here.

2.5.1 Hybrid Automata Stochastic Language

The Hybrid Automata Stochastic Language [10] (HASL) is a temporal logic formalism for the verification of stochastic models. As illustrated in Figure 2.11, it enables the assessment of sophisticated performance indicators of sWN models S, through the specification of a property $\varphi \equiv (\mathcal{A}_S, Z)$ formally defined by the combination of a Linear Hybrid Automaton (LHA) $\mathcal{A}_S$ and a target expression Z. The functioning of the framework can be summarised as follows:

1. A sufficiently large number of finite traces are sampled by simulation from S and synchronised on-the-fly with $\mathcal{A}_S$.
2. Those traces that satisfy the acceptance conditions of $\mathcal{A}_S$ are retained, and the statistics collected in the variables of $\mathcal{A}_S$ are then exploited to construct an ϵ-confidence level estimate (with confidence interval width δ) of the quantitative measure of interest Z.

Synchronisation of an sWN model with an LHA. The first component of the property φ is the LHA $\mathcal{A}_S$. Linear hybrid automata (LHA) were first introduced by Thomas Henzinger in [36] as a restricted class of hybrid automata where linear differential inclusions govern the evolution of continuous variables. This restriction makes them amenable to symbolic analysis while still being expressive enough to capture a wide range of real-life systems. The theory and formalism of LHA were later exploited in [10], where they were synchronised with a generalised stochastic Petri net to perform a wide variety of performance analyses and property verifications through a simulation procedure. In this setting, the LHA acts as an observer that guides the simulation of the net by storing and updating variables, and by determining both the conditions under which the simulation terminates and whether a given trajectory is considered or discarded.

Definition 2.16 (Linear Hybrid Automaton) A Linear Hybrid Automaton (LHA), synchronised with an sWN $S = (P, T, F, W, \Sigma_S, \lambda, m_0)$, is a tuple $\mathcal{A}_S = (Ev, Loc, Init, Acc, X, flow, \Lambda, \rightarrow)$ where:

- $Ev = \Sigma_S \cup \{\tau\}$ is the alphabet of observed events,
- Loc is a finite set of locations,
- $Init \subseteq Loc$ is a singleton set containing the unique initial location,
- $Acc \subseteq Loc$ is the set of accepting locations,
- $X = \{x_1, \ldots, x_n\}$ is a finite set of real-valued variables,
- $flow : Loc \rightarrow (R_s(S) \rightarrow \mathbb{R}^n)$ specifies, for each location, the rate (i.e., first derivative) with which each variable x_i evolves depending on the current marking of S,
- $\Lambda : Loc \rightarrow (R_s(S) \rightarrow \mathbb{B})$ assigns to each location an invariant (i.e., a Boolean predicate depending on the current marking of S), and
- $\rightarrow$ is a set of transitions of the form

$$l \xrightarrow{Ev', \gamma, U} l'$$

  where
  - γ is an enabling guard (an inequality over the variables X),
  - $Ev' \subseteq Ev$ is either a set of event names (the transition is synchronously traversed upon the occurrence of an event in Ev') or $\#$ (the transition is autonomously traversed without synchronisation), and
  - U is the set of variable updates.

Example 2.9 (Synchronisation between an LHA and an sWN).

Figure 2.12 shows an example of an sWN S_2 synchronized with an LHA $\mathcal{A}_{S_2}$, as depicted in Figure 2.13.

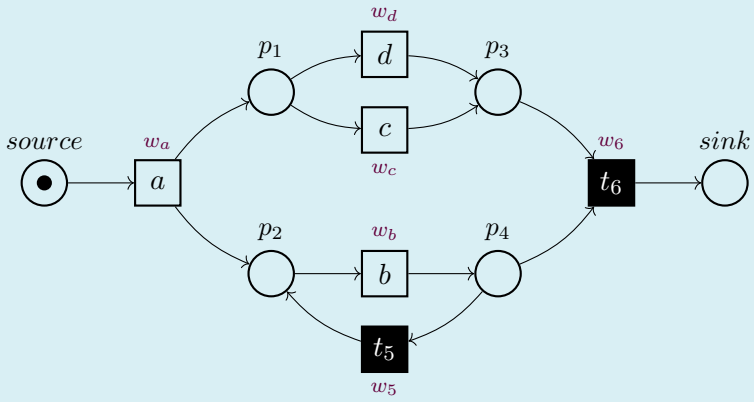


Figure 2.12 – sWN S_2

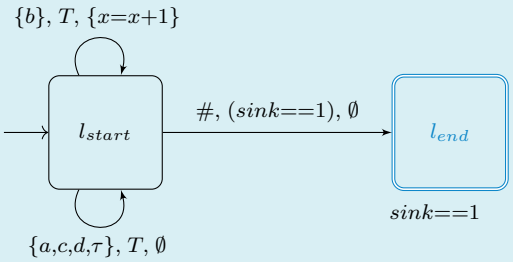


Figure 2.13 – LHA $\mathcal{A}_{S_2}$ synchronized with S_2

The net S_2 contains four labelled and two silent transitions. The automaton $\mathcal{A}_{S_2}$ has an initial location l_{start} and an accepting location l_{end}, and uses a discrete variable x to count how many times the transition b is fired. Synchronisation works as follows:

$$l_{start} \xrightarrow{\{b\},\, \top,\, \{x=x+1\}} l_{start} \qquad \text{and} \qquad l_{start} \xrightarrow{\{a,c,d,\tau\},\, \top,\, \emptyset} l_{start}$$

Finally, the automaton moves to the accepting state when the sink place of S_1 is marked:

$$l_{start} \xrightarrow{\#,\, (sink==1),\, \emptyset} l_{end}$$

This example shows how an automaton can be used to monitor discrete aspects of the behaviour of a stochastic workflow net. In particular, the counter x records the number of times transition b is executed before the final marking is reached. Once the synchronised execution terminates, these values can be exploited to verify probabilistic properties of the model.

Within an HASL model checker, the engine launches several simulations over both the sWN S and the LHA $\mathcal{A}_S$, starting respectively from the initial marking m_0 of S and the initial location $l_0 \in Init$ of $\mathcal{A}_S$. Each simulation may terminate either by

accepting the randomly generated trace (if the LHA reaches an accepting location), or by rejecting it (if the model falls into a deadlock, a livelock, or if the simulation runs for too long, as determined by the model checker). The model checker also determines when a sufficient number of traces have been generated to provide a statistically precise analysis. Finally, the values of the variables X of the LHA in the accepting traces are passed to the second component of φ.

Target expression. The second component of φ is an expression Z resulting from the following grammar over the variables of the considered LHA $\mathcal{A}_S$:

$$\begin{aligned} Z &::= AVG(Y) \mid Z + Z \mid Z - Z \mid Z \times Z \mid Z/Z \mid Pdist \\ Pdist &::= PDF(Y, step, start, stop) \mid CDF(Y, step, start, stop) \mid PROB() \\ Y &::= c \mid Y + Y \mid Y - Y \mid Y \times Y \mid Y/Y \mid last(y) \mid min(y) \mid max(y) \mid avg(y) \\ y &::= c \mid y + y \mid y - y \mid y \times y \mid y/y \end{aligned}$$

where:

- $c \in \mathbb{R}$ is a constant.
- $y \in X$ is a variable of $\mathcal{A}_S$.
- $last(y)$ is the value that y has on the accepted trace at the end of synchronisation.
- $min(y)$ is the minimum value that y has taken along the accepted trace.
- $max(y)$ is the maximum value that y has taken along the accepted trace.
- $avg(y)$ is the mean value measured for y along the accepted trace.
- $PROB()$ is the ratio of accepted traces over the total number of traces simulated
- $AVG(Y)$ is the mean value (i.e. the centre of the confidence interval) of Y.
- $PDF(Y, step, start, stop)$ is the approximation of the probability distribution function for Y, resulting by using $[start, stop]$ as discretised support for the distribution (i.e. $[start, stop]$ is split in $(stop - start)/step$ equally sized buckets of length $step$).
- $CDF(Y, step, start, stop)$ is the approximation of the cumulative distribution function for Y, resulting by using $[start, stop]$ as discretised support for the distribution (i.e. $[start, stop]$ is split in $(stop - start)/step$ equally sized buckets of length $step$).

In practice, it offers a wide range of analysis possibilities over the model S, which can significantly assist in evaluating and understanding its behaviour.

Example 2.10 (HASL target measures).

To illustrate, we present a few instances of Z expressions referring to the LHA $\mathcal{A}_{S_1}$ and the corresponding sWN model S_1 from Example 2.5.1:

- $Z_1 = AVG(last(x))$ is the average number of times transition b is exe-

cuted before reaching the final marking.

- $Z_2 = PDF(last(x), 1, 0, 5)$ is the probability distribution of the number of firings of transition b, approximated on the support $[0, 5]$ with buckets of width 1.
- $Z_3 = PROB()$ is the probability of reaching the accepting marking, which can be used to detect whether the sWN is unsound.
- $Z_4 = AVG(max(x) - min(x))$ is the average variation (spread) of the counter x along an accepted trace.

2.5.2 Approximate Bayesian Computation

Approximate Bayesian Computation [58, 70] (ABC) methods aim at estimating the posterior distribution of a model's parameters θ given some observed (experimental) data y_{exp}. Let $\pi(\theta)$ denote a prior distribution on the parameters, $y_{exp} \in \mathcal{Y}$ the observations, and $p(y|\theta)$ the likelihood function of the model. The objective of Bayesian inference is to determine the posterior distribution:

$$\pi(\theta|y_{exp}) = \frac{p(y_{exp}|\theta)\,\pi(\theta)}{\int p(y_{exp}|\theta')\,\pi(\theta')\,d\theta'}. \tag{2.3}$$

In most models, the likelihood function $p(y_{exp}|\theta)$ is either too expensive to compute or even intractable, thus hindering the exact determination of the posterior distribution. ABC algorithms address this limitation by providing an approximation, denoted $\pi_{ABC,\epsilon}$ (with $\epsilon \in \mathbb{R}_+^*$ a tolerance value), of the posterior $\pi(\theta|y_{exp})$ that can be made arbitrarily precise.

Rejection sampling. In its simplest form, known as rejection sampling (Algorithm 1), ABC consists of a straightforward iterative procedure: candidate parameter values θ' are repeatedly sampled from the prior distribution $\pi(\theta)$, and for each sample a synthetic dataset $y' \sim p(\cdot|\theta')$ is generated from the model. The parameter θ' is accepted if the distance between the summary statistics of y' and those of the observed data y_{exp} is below the tolerance threshold ϵ, that is, if $\rho(\eta(y'), \eta(y_{exp})) \leq \epsilon$.

Although effective, ABC rejection sampling converges slowly, especially for small tolerance values ϵ. To overcome this limitation, the Sequential Monte Carlo extension of ABC, called ABC-SMC [13], was introduced. It accelerates convergence by progressively refining the parameter search through a sequence of decreasing tolerance levels, thereby improving the efficiency of the approximation process.

Sequential Monte Carlo. ABC-SMC extends the basic rejection sampling approach by introducing a sequence of intermediate populations of particles (parameter samples), each associated with a decreasing tolerance value ϵ_t, for $t = 1, \ldots, T$, where T denotes the number of layers in the sequential procedure. The algorithm starts with a large tolerance ϵ_1, allowing many parameter samples to be accepted,

Algorithm 1 ABC rejection sampling

Require: y_{obs} (observations), ϵ (tolerance),
ρ (distance metric), η (summary statistics)
Ensure: $\{\theta_i\}_{i=1}^n$ drawn from $\pi_{ABC,\epsilon}$
for $i = 1 : n$ **do**
repeat
$\theta' \sim \pi(.)$
$y' \sim p(.|\theta')$
until $\rho(\eta(y'), \eta(y_{obs})) \leq \epsilon$
$\theta_i \leftarrow \theta'$
end for
return $\{\theta_i\}_{i=1}^n$ at tolerance ϵ

and gradually reduces the tolerance level until the target ϵ_T is reached. At each stage, the accepted particles are resampled and perturbed, typically using a Markov kernel $K_t(\cdot|\theta)$, to generate a new candidate population that is more concentrated around promising regions of the parameter space. Weights are assigned to particles to correct for the bias introduced by resampling and perturbation, ensuring that each intermediate distribution adequately approximates the posterior restricted by $\rho(\eta(y'), \eta(y_{exp})) \leq \epsilon_t$.

Formally, if $\{\theta_i^{(t-1)}, w_i^{(t-1)}\}_{i=1}^N$ denotes the weighted population of parameters at iteration $t-1$, then a new candidate θ^* is generated by sampling from this population according to the weights $w_i^{(t-1)}$ and perturbing the chosen particle via K_t. A simulated dataset $y^* \sim p(\cdot|\theta^*)$ is produced, and θ^* is accepted if $\rho(\eta(y^*), \eta(y_{exp})) \leq \epsilon_t$. The weight of an accepted particle θ^* is then updated as:

$$w^{(t)}(\theta^*) = \frac{\pi(\theta^*)}{\sum_{i=1}^N w_i^{(t-1)} K_t(\theta^*|\theta_i^{(t-1)})}.$$

This sequential approach offers several advantages over plain rejection sampling. First, it reduces the number of simulations required to reach trim tolerance levels, as the search is progressively focused on high-probability regions of the parameter space. Second, it naturally provides an adaptive exploration mechanism: the distribution of particles evolves according to the observed data and the prior. Finally, ABC-SMC yields not only a point estimate of the parameters but an approximation of the full posterior distribution, enabling richer statistical analyses.

Chapter 3

Exact Computation of the Stochastic Workflow Net Language

Stochastic workflow nets (sWNs) are arguably the most common modelling formalisms used to account for the stochastic dimension of observed processes [31]. In essence, an sWN is a workflow net (WN) enriched with a vector of positive real-valued weight parameters ($\overline{w} \in \mathbb{R}_{>0}^{|T|}$) whose values are used to determine the probability with which each transition, enabled in a given marking, fires. In the context of sWN modelling, the process discovery problem consists of deriving an sWN model that reproduces as closely as possible both the control-flow as well as the stochastic dimension of the observed process, relying on a stochastic resemblance criterion [52, 46] to drive the discovery. In this context, stochastic resemblance boils down to assessing the similarity between the stochastic language of the log and that of the sWN. If the log gives the stochastic language of the observed process, the computation of the stochastic language of an sWN model is non-trivial. Because a usually considerable number of silent transitions co-occur with labelled transitions in multiple control-flow constructs, computing the probability of a trace in an sWN is non-trivial: a single trace can be induced by a potentially large set of transition firing sequences [51].

In this chapter, we introduce a method for determining the exact stochastic language issued by an sWN model via breadth-first unfolding of the underlying (probabilistic) reachability graph (RG). In order to correctly identify the probability of each sequence of actions generated by an sWN, the procedure constructs on the fly a dedicated directed acyclic graph (DAG) whose nodes store information about the traces (sequences of actions) emitted by the sWN along the corresponding unfolded path (i.e., the path traversed so far in the RG).

In order to deal with infiniteness (i.e., the presence of loops in the sWN reachability graph), termination of the unfolding can be controlled via different criteria, for example, by imposing a bound on the length of the unfolded traces, or by means of

a bound on the total probability mass of the unfolded traces. For practical reasons, however, we opted, in our implementation, to let the unfolding process continue, ensuring that every trace of the corresponding event log (i.e., the event log from which the sWN has been discovered) is discovered via unfolding. Notice that, even in this case, termination is guaranteed as long as the process discovery algorithm enjoys perfect fitness (i.e. every trace of the log is guaranteed to be part of the sWN language). The rest of this chapter is organised as follows:

- Section 3.1 discusses the challenges involved in designing such a procedure and introduces the intuition behind the algorithm through a simple illustrative example.
- Section 3.2 presents the algorithm for computing the stochastic language of an sWN. The method relies on a breadth-first unfolding of the reachability graph, enabling the computation of trace probabilities given a fixed weight vector.
- Section 3.3 describes a more efficient implementation of this unfolding strategy. By dynamically generating reusable functions and employing memoisation, the computational cost of repeated evaluations is significantly reduced, which is especially beneficial in an optimisation procedure.
- Section 3.4 describes a major limitation of unfolding-based strategies for computing sWN stochastic languages: namely, the presence of loops consisting solely of silent transitions.
- Section 3.5 concludes the chapter by summarising the advantages and limitations of the proposed method and discussing its integration within an optimisation framework.

3.1 A motivating example

To illustrate the procedure for the exact computation of an sWN language intuitively, let us consider a simple example. Let L_1 be an artificial event log with corresponding stochastic language

$$\mathcal{S}_{L_1} = [\langle a, b, c\rangle^{0.3}, \langle a, d, b\rangle^{0.2}, \langle a, c, b, b\rangle^{0.2}, \langle a, b, c, b\rangle^{0.2}, \langle a, b, b, d\rangle^{0.1}].$$

Notice that each unique trace of L_1 (i.e. each trace in $\mathcal{T}_{\mathcal{S}_{L_1}}$) begins with the activity a and that, after such common prefix, activity b may be repeated once and may occur either before or after activity c or d, which however occur mutually exclusively in a trace. In other words, this means that the structure of the traces exhibits a combination of concurrency (different interleaving) between a conflict (mutually exclusive choice) and a loop (repetition).

Figure 3.1 shows an sWN model S_1 (i.e., a WN enriched with real-valued, positive weights) that achieves perfect fitness w.r.t. log L_1 (i.e. all traces of L_1 are in the language of the underlying WN N_1, $\mathcal{T}_{S_{L_1}} \subseteq \mathcal{L}_{N_1}$). Observe that in order to model the repetition of b, the net includes a loop pattern, which makes the net's language

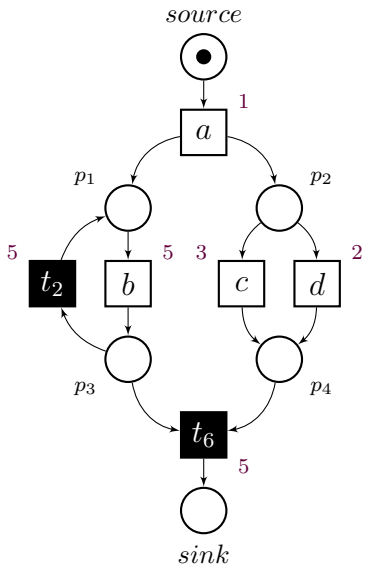


Figure 3.1 – Stochastic workflow net S_1 discovered from log L_1

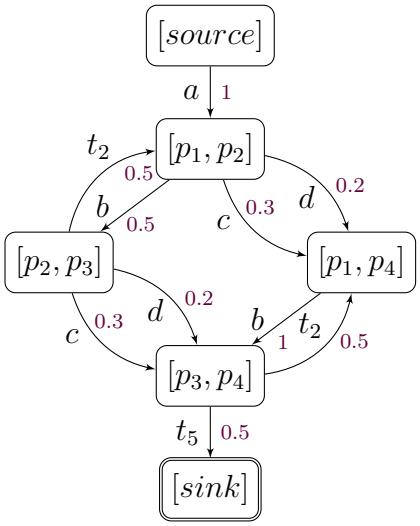


Figure 3.2 – Reachability graph $R_g(S_1)$ annotated with transition probabilities

infinite by allowing any word with an arbitrary repetition of b. The rest of the net accurately models the fact that every sequence starts with a, and that after this initial transition, the process splits into two branches: one containing the loop over b, and the other representing the conflict between c and d, thus capturing the overall concurrency.

Figure 3.2, instead, depicts the "probabilistic" reachability graph of S_1, that is, the reachability graph of the underlying WN enriched with transition probabilities resulting from the weights of the transitions of S_1. In any given state of the RG, the probability value of an outgoing transition is given by the ratio of the transition weight over the sum of the weights of transitions that are concurrently enabled. Therefore, for example, in the initial state $[source]$ transition a occurs with probability 1 (as it is the only transition enabled) while in marking $[p_1, p_2]$, the probability of the three concurrently enabled transitions, i.e., b, c and d, is:

$$\mathbb{P}(b|[p_1,p_2]) = \frac{W(b)}{W(b)+W(c)+W(d)} = \frac{5}{10} = 0.5$$

$$\mathbb{P}(c|[p_1,p_2]) = \frac{W(c)}{W(b)+W(c)+W(d)} = \frac{3}{10} = 0.3$$

$$\mathbb{P}(d|[p_1,p_2]) = \frac{W(d)}{W(b)+W(c)+W(d)} = \frac{2}{10} = 0.2$$

Identification of the sWN language via traversal of the RG. The procedure for determining the stochastic language of an sWN model (formalised in Algorithm 2)

consists of the construction *on the fly* of a dedicated directed acyclic graph (DAG) whose nodes are labelled with the "unfolded" traces (sequences of transition labels) and corresponding trace probability. The construction of such a DAG is obtained by breadth-first traversal of the corresponding probabilistic reachability graph. Notice that since the RG may contain loops, the traversal may not terminate (leading to infinite depth of the DAG). On the other hand, because of the structural characteristics of WNs, it is guaranteed that such a DAG 1) contains at least one leaf node (corresponding to reaching of the RG deadlock state $[sink]$ and 2) every node lies on a path from the root $[source]$ to some leaf node $[sink]$.

Figure 3.3 depicts the DAG resulting from "unfolding" of the probabilistic reachability graph $R_g(S_1)$ (Figure 3.2). Nodes are arranged by *level*, corresponding to the length of the path being unfolded (i.e., the number of transitions fired). Each node in the DAG is annotated with three pieces of information:

- the marking corresponding to the currently reached RG state.
- the trace prefix formed by the sequence of labels of non-silent transitions fired along the path from the initial state to the current node,
- the probability of reaching the node according to the considered parameterisation W of the model.

In Figure 3.3, leaf nodes (corresponding to the $[sink]$ marking) are depicted within double rectangles. Notice that traces leading to these nodes, together with the associated probabilities, form part of the stochastic language $\mathcal{S}_{S_1}$ of the net S_1. Leaf nodes depicted in blue correspond to traces that are part of the initial log stochastic language $\mathcal{S}_{L_1}$.

Referring to the DAG in Figure 3.3, observe that the central path corresponds to a firing sequence where activity b occurs, resulting in an arbitrary number of alternated occurrences of transition b and t_2. The upper part of the DAG corresponds to the behaviour where activity c is chosen in the conflict against d, while the lower part corresponds to the behaviour where d is chosen. As a result, levels 3 and 4 contain all possible interleavings between one occurrence of b and c for the upper part ($\langle a, b, c\rangle$ and $\langle a, c, b\rangle$), and between one occurrence of b and d for the lower part ($\langle a, b, d\rangle$ and $\langle a, d, b\rangle$). Similarly, levels 5 and 6 contain all possible interleavings between two occurrences of b and c (or d):

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

Of course, every trace begins with activity a, which is added by the unique node at level 1 and is part of every execution path of the DAG.

For each node in the DAG, the probability of reaching that node, and thus obtaining the corresponding trace, is computed incrementally during the exploration. At each step, the probability of a node is obtained by multiplying the probability of its parent node by the probability of the transition taken to reach it. Note that multiple paths can lead to the same trace in the same marking, hence a DAG node may be on several paths connecting the root with one leaf. If multiple paths end in

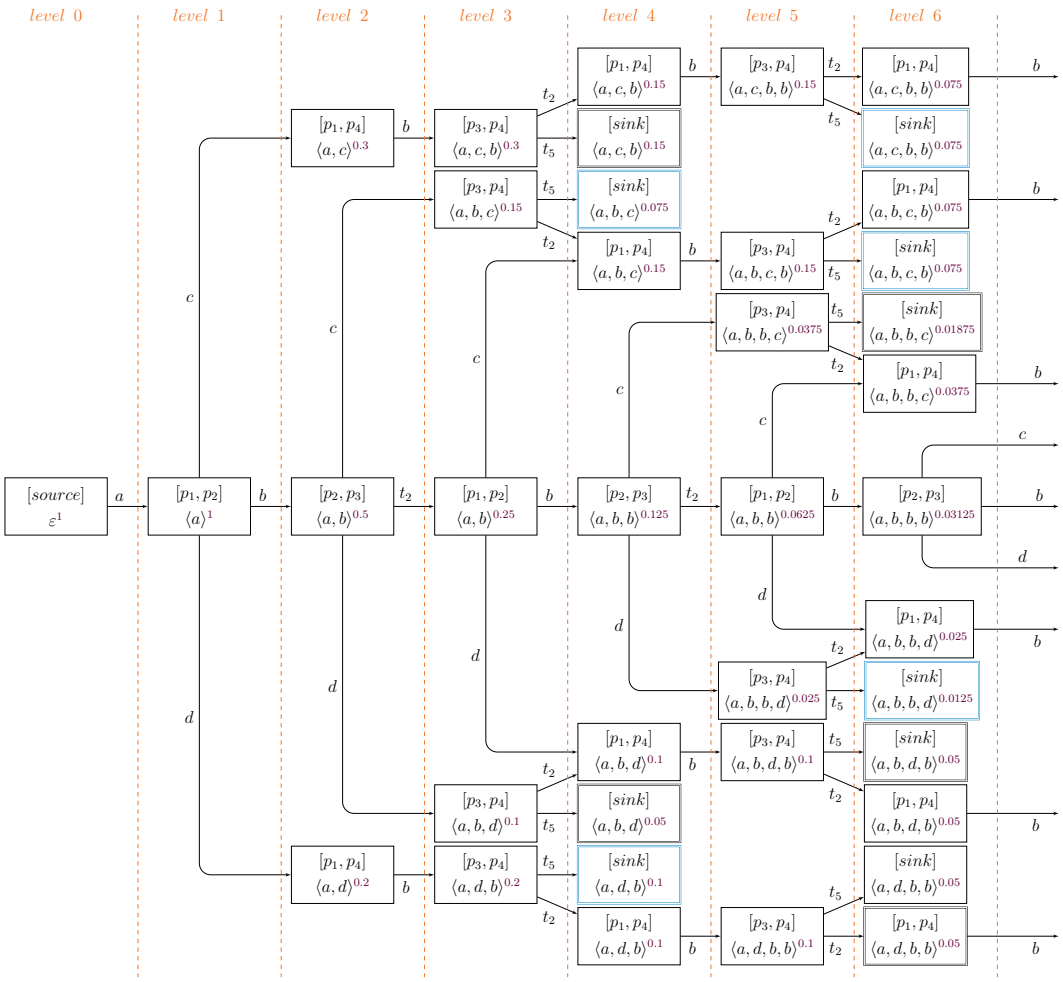


Figure 3.3 – Enumeration of all possible paths of length 6 in S_1

the same node, the corresponding probability is obtained by summing the probabilities of reaching the node through all possible paths. This ensures that, for every trace prefix represented in the DAG, its associated probability accurately reflects the likelihood of following that execution path in the sWN.

Due to the presence of silent transitions, which are very common and often have a high likelihood of being added to the model to match log patterns, a transition firing can change the marking of the net without modifying the trace. Such paths have different probabilities, since the sequences of transitions that lead to them are distinct. However, this can lead to multiple cases that must be taken into account when unfolding the reachability graph of the model:

- **Same trace at different levels.** This occurs when, at different levels (consecutive or not), we encounter the same trace. In the example, this happens every time t_2 or t_5 is fired. This case requires taking both the level and the marking into account when identifying a given node. Otherwise, by mentioning only the trace, we would be unable to determine the length of the execution path or the actual state of the net.
- **Same marking, different traces at the same level.** This can result from firing the same number of transitions but in a different order. For instance, at level 4 (Figure 3.3), we can reach $[p_1, p_4]$ or $[sink]$ through different execution paths. This case requires the same solution as the previous one, with the same level of precision in node identification.
- **Same marking and same trace at the same level.** This is the most delicate case to address. It means that different paths, with the same path length, lead to precisely the same node in the DAG. This situation appears several times in the DAG in Figure 3.3. For instance, at level 4, there are two possible execution paths that both reach marking $[p_1, p_4]$ emitting trace $\langle a, b, c\rangle$. The two paths diverge earlier: one executes c immediately after $\langle a, b\rangle$, while the other first resets the loop by firing silent transition t_2 and then executes c. As a result, we have two execution paths leading to the same node, possibly with different probabilities. Therefore, the probability of such a merged node depends on both parent nodes and is computed as the sum of their contributions:

$$\big(\overbrace{0.15}^{\text{prob. of parent 1}} \times \underbrace{0.5}_{\text{prob. of firing } t_2}\big) + \big(\overbrace{0.25}^{\text{prob. of parent 2}} \times \underbrace{0.3}_{\text{prob. of firing } c}\big) = 0.15$$

  Before exploring the descendants of such a node, we must ensure that all possible paths leading to it have been accounted for. Otherwise, its probability will be incomplete, and this incompleteness will propagate to subsequent nodes.

Another important thing to notice is that the unfolding of the reachability graph $R_g(S_1)$ leads to an infinite number of paths. Looking at $R_g(S_1)$, we notice two arcs that allow us to return to an already visited marking: both corresponding to the loop over b. This means that the reachability graph contains cycles, which is why its unfolding into a DAG results in infinitely many paths. Each time the loop transition

is taken, new paths are created in the DAG, all corresponding to an additional occurrence of b in the net traces. As a result, the stochastic language of the net is infinite, since there is no upper bound on the number of times b can be repeated before reaching the final marking. In the example, at level 3, we have already encountered an occurrence of b. In some paths, we also encounter the only occurrence of c, and in others, the only occurrence of d. When moving to level 4, we can either reach the final marking and end those paths, or loop again over b, or execute c or d for paths that have not yet encountered them. A response to this problem of explosion and infiniteness is to decide when to stop the exploration and which paths deserve to be explored. One efficient approach is to restrict the search to traces that appear in the log and terminate the procedure once all such traces have been identified. However, this requires ensuring that the model under study has perfect fitness with respect to the log. Then, based on the list of all prefixes formed from the set of traces in the log, we can determine, for each node, whether further exploration from this node will lead to traces that are present in the log or not.

Note that, in Figure 3.3, we have explored every possible path for the first seven levels, including those that do not yield traces from the log. At each level, the sum of the probabilities of all nodes is equal to 1 minus the total probability mass already absorbed by sink nodes at earlier levels. In other words, the probability mass is conserved across the unfolding, with the portion corresponding to completed executions being removed from the active part of the graph. From level 0 to level 4, the sum is indeed equal to 1. However, at level 4, there are four execution paths in the $[sink]$ marking, whose total probability mass is 0.375. Therefore, in the following level, the remaining active probability mass is $1 - 0.375 = 0.625$.

In summary, the basic aspects needed in order for the stochastic language unfolding procedure to work efficiently are as follows:

- We require three pieces of information to ensure that every node is unique and cannot be confused with another: the level, which is the length of the execution path; the trace, obtained from the labelling of the execution path; and the actual marking of the net.
- Before exploring the descendants of a given node, we must ensure that every possible path leading to this node has been taken into account.
- Finally, to achieve efficiency and handle infinite languages, we restrict the search to only those traces and trace prefixes that are present in the log.

3.2 Computation of the stochastic language of an sWN via log-driven unfolding

The formalisation of the log-driven procedure for computing the stochastic language of an sWN model S via breadth-first traversal of the underlying (probabilistic) reachability graph is given in terms of the function $\textsc{UnfoldRG}(\mathcal{T}, S, R_g(S))$ whose computational definition is given in Algorithm 2). $\textsc{UnfoldRG}(\mathcal{T}, S, R_g(S))$ takes

set of traces $\mathcal{T} \subseteq \Sigma^*$ over an alphabet of activities Σ (i.e. the event log), an sWN $S = (P, T, F, W, \Sigma, \lambda, m_0)$, and its reachability graph $R_g(S) = (R_s(S), A)$ as inputs and returns a set of pairs $(tr, pr) \in \mathcal{T} \times [0, 1]$, where pr is the probability of trace tr in the stochastic language $\mathcal{S}_S$, i.e., $pr = \mathcal{S}_S(tr, \overline{w})$. In the following, we describe the functioning of the algorithm, block by block.

Lines 1 - 8. The algorithm begins by annotating each arc of the reachability graph $R_g(S)$ with the probability associated with the corresponding transition firing, as determined by the weight function of the sWN. Specifically, we iterate over every arc in the reachability graph, where each arc represents the firing of a transition t from a source state st to a target state st'. For each such arc (st, st', t), we compute the firing probability of transition t in state st according to the stochastic semantics of the model. This probability is given by the weight $W(t)$ divided by the sum of the weights of all transitions enabled in st, denoted by the variable sum_w. Computing it involves an inner loop over all arcs originating from the same source state st, in order to accumulate the weights of all transitions enabled in that state.

Note that the reachability graph $R_g(S)$ could be computed from S inside UnfoldRG. We include it as an input parameter because, in certain use cases, the stochastic language may be computed multiple times with varying transition weights only. These variations affect only the probabilities assigned to the arcs, not the structure of the reachability graph. Therefore, it is sufficient and more efficient to compute the reachability graph only once before invoking UnfoldRG multiple times.

Lines 9 - 11. The probabilities of the traces in $\mathcal{T}$ are stored in the set D, which contains pairs (tr, pr), where tr is a trace and pr is a real number representing its probability in the stochastic language. To follow a breadth-first unfolding of $R_g(S)$, we use a queue Q. The elements of Q are nodes of the directed acyclic graph (DAG) constructed during the unfolding and are represented by pairs of the form $((st, level, tr), pr)$, where:

- the first component is a triple $(st, level, tr)$:
  - $st \in R_s(S)$ is a state (marking) of S;
  - $level \in \mathbb{N}$ is the length of the unfolded path that leads to st (from the initial state m_0 of S);
  - $tr \in \textit{prefixes}(\mathcal{T})$ is a prefix of some trace in $\mathcal{T}$;
- the second component $pr \in [0, 1]$ is the probability of the execution path.

The queue Q is initialised by enqueuing $((m_0, 0, \varepsilon), 1)$, where the triple corresponds to the initial marking reached by an unfolded path of length 0 with empty trace, and the probability 1 indicates that this node is part of every execution. For both the set D and the queue Q, we assume the existence of a procedure InsertOrIncrease(key k, value v), which inserts key k with value v if k is not yet present, and otherwise increases the value associated with k by v. In the case of D, the key is a trace, whereas in the case of Q, the key is a triple $(state, level, trace)$. It is important to note that the $level$ is necessary because, due to silent transitions, the same $(state, trace)$ pair can sometimes be reached by firing either k or $k + 1$ transitions.

Algorithm 2 UnfoldRG – Reachability Graph Unfolding

Require: A set of traces $\mathcal{T} \subseteq \Sigma^*$,
a stochastic workflow net $S = (P, T, F, W, \Sigma, \lambda, m_0)$,
and its reachability graph $R_g(S) = (R_s(S), A)$
Ensure: A set $D \subseteq \mathcal{T} \times [0, 1]$ of pairs (tr, pr) where $pr = \mathcal{S}_S(tr, \overline{w})$

$PrArcs \leftarrow \{\}$
for all $(st, st', t) \in A$ **do**
  $sum\_w = 0$
  for all $(st, st'', t') \in A$ **do**
    $sum\_w \leftarrow sum\_w + W(t')$
  end for
  $PrArcs \leftarrow PrArcs \cup \{(st, st', t, \frac{W(t)}{sum\_w})\}$
end for

$D \leftarrow \{\}$
$Q \leftarrow$ empty queue
Q.ENQUEUE$((m_0, 0, \epsilon), 1)$

while Q is not empty **do**
  $((st, level, tr), pr) \leftarrow Q$.DEQUEUE$(\,)$
  for all $(st, st', t, prarc) \in PrArcs$ **do**
    $tr' \leftarrow tr + \lambda(t)$
    $pr' \leftarrow pr \cdot prarc$
    if st' is $[sink]$ **then**
      if tr' is a trace in $\mathcal{T}$ **then**
        D.INSERTORINCREASE(tr', pr')
      end if
    else if $tr' \in prefixes(\mathcal{T})$ **then**
      Q.INSERTORINCREASE$((st', level + 1, tr'), pr')$
    end if
  end for
end while

for all $s \in \mathcal{T}$ **do**
  if there is not such pair $(tr, pr) \in D$ that $tr = s$ **then**
    D.INSERTORINCREASE$(s, 0)$
  end if
end for

return D

To maintain the breadth-first order, these sequences must be distinguished.

Lines 12 - 25. The unfolding proceeds by dequeuing an element from Q (line 13) and exploring all outgoing arcs of the corresponding state in the reachability graph (line 14). For each arc, line 15 updates the trace by appending the activity associated with the fired transition, and updates the probability by multiplying it by the probability of the arc. The resulting trace is stored in tr', while the resulting probability is stored in pr'. From line 17 to 23, the algorithm handles the case in which the newly reached state is the final marking $[sink]$. If the generated trace is present in $\mathcal{T}$, the pair (tr', pr') is either inserted into D or used to increase the probability of the existing entry. If the newly reached state is not final but the generated trace is a prefix of some trace in $\mathcal{T}$ (line 21), we either enqueue a new item or increase the probability associated with an existing one.

Lines 26 - 30. Once the unfolding-based generation of the traces emitted by model S has terminated we add to the output set D any trace of the input set $\mathcal{T}$ that has not been unfolded from S, associating it with probability 0. Notice that, in the case of a perfectly fitting model S, this step is redundant.

Algorithm 2 is presented in such a way that the unfolding is restricted to a set of traces, which is helpful to evaluate traces present in an event log. This is an advantageous choice from the point of view of execution time since event logs usually contain much fewer traces than the number of traces their mined workflow net counterpart can produce (discovered WN have likely below one precision). Moreover, most mined workflow nets contain cycles leading to an infinite number of possible traces. It is straightforward to modify the algorithm to make the calculations independent of a set of traces. Suppose the number of possible traces is infinite. In that case, one can put a limit on the length of the traces (disregarding silent transitions) or cover a given amount of probability, for example, to stop the unfolding when the total probability in D is more than a user-defined threshold (0.9, for instance). Due to the complexity of the workflow net or the characteristics of the event log, even restricting the calculation to the traces present in the event log makes the execution time unfeasible. This can easily occur in the presence of long traces that various transition sequences of the workflow net can generate. In this case, a limit on the length of the traces can be introduced to obtain at least an approximation.

3.2.1 A more complex example

To illustrate Algorithm 2, we use an example based on an artificial log L_2 that contains longer and more intricate traces than the previous example L_1, resulting in a more profound and more complex unfolding DAG. The stochastic language of this log is:

$$\mathcal{S}_{L_2} = [\langle a, a\rangle^{0.3}, \langle a, a, a\rangle^{0.2}, \langle a, a, a, a\rangle^{0.1}, \langle b, a, b, a, b\rangle^{0.2}, \langle a, a, b, b, a\rangle^{0.2}]$$

The set of traces $\mathcal{T}_{S_{L_2}}$ from L_2 involves only two activities: a and b. These activities appear multiple times and exhibit both repetition (looping behaviour) and inter-

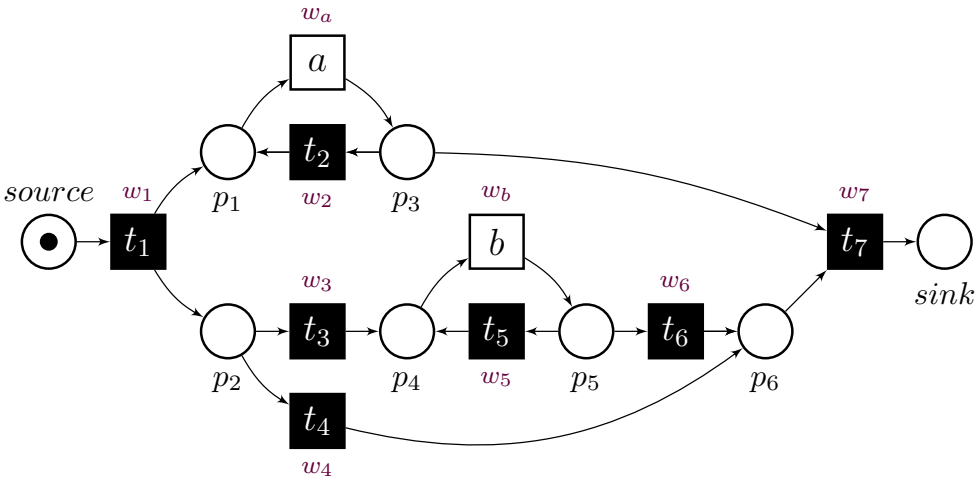


Figure 3.4 – Stochastic workflow net S_2 discovered from log L_2

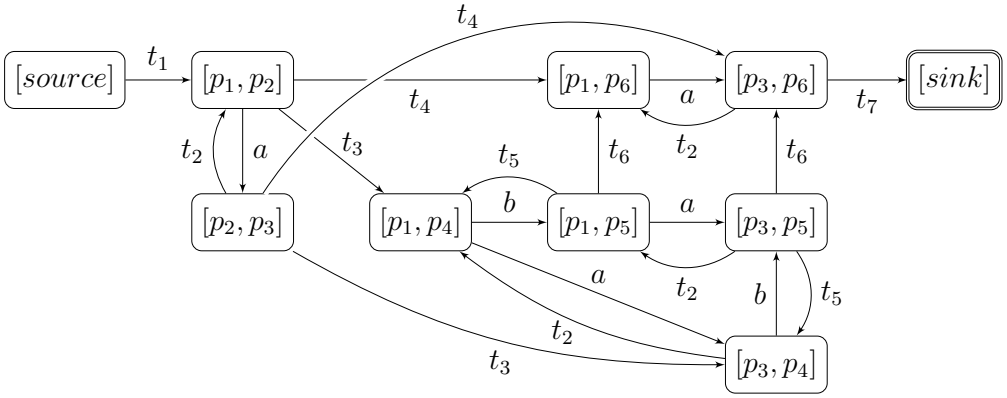


Figure 3.5 – Reachability graph $R_g(S_2)$ annotated with transition probabilities

leaving. Some traces contain consecutive repetitions of a, while others alternate between a and b in various orders and nesting patterns.

The sWN shown in Figure 3.4, obtained using the Inductive Miner [49], is capable of reproducing all the traces present in the log L_2. It captures a concurrent pattern involving two loops: one over activity a, and another over activity b. The concurrency allows for interleaved repetitions of both activities, reflecting the structure observed in the log. Notably, some traces do not contain any occurrence of activity b. To account for this, the model includes a choice between entering the loop over b or skipping it entirely. The sWN contains seven silent transitions and two transitions that are labelled with the two activities. Unlike the previous example, transition weights are not specified here, in order to emphasise the general behaviour of the algorithm rather than a specific numerical instance.

The corresponding reachability graph is shown in Figure 3.5. For the sake of readability, the probabilities associated with transitions between markings are not shown in the figure. Note that the language associated with the sWN is not finite since there are loops containing non-silent transitions.

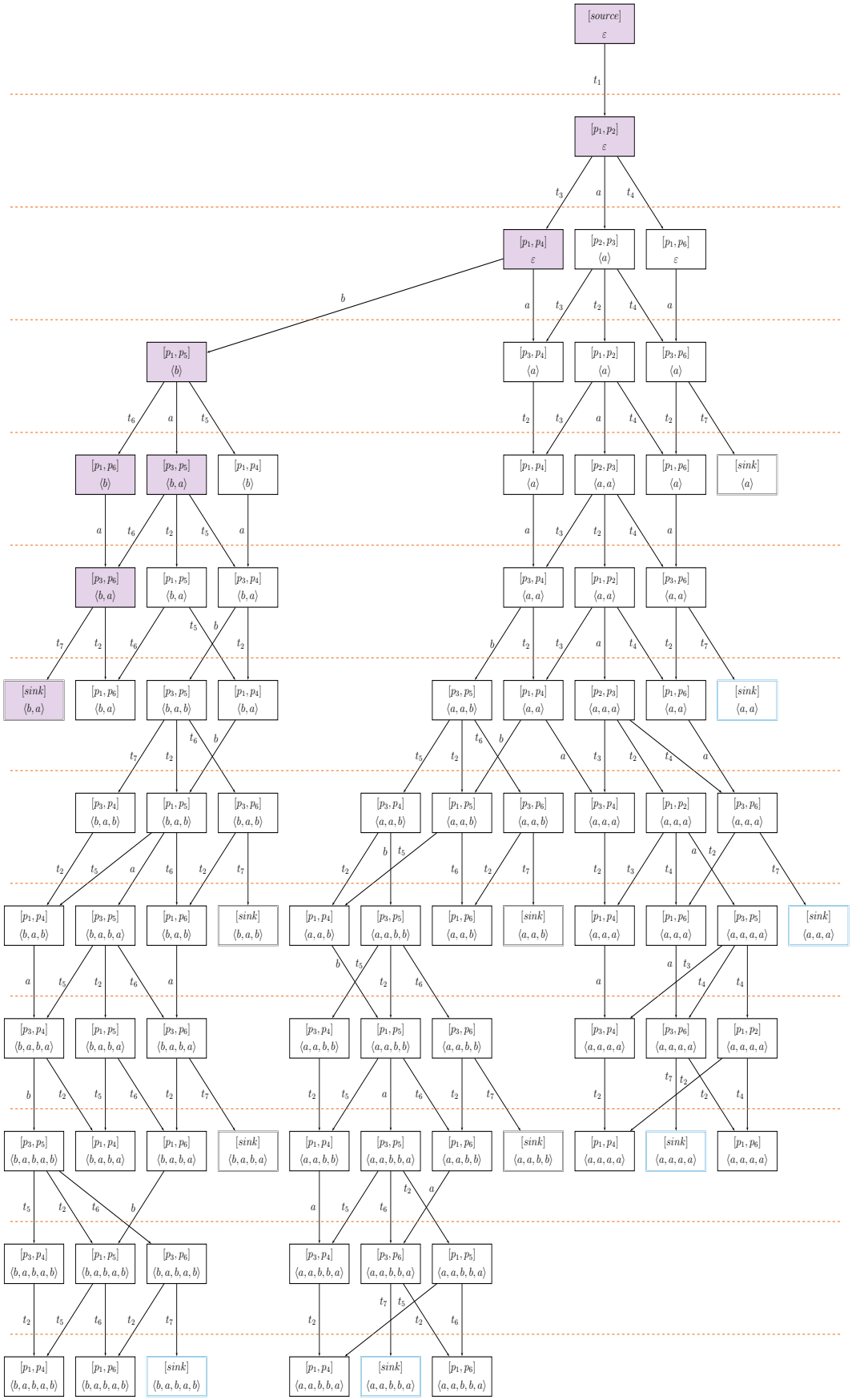


Figure 3.6 – Unfolding DAG of $R_g(S_2)$ based on Algorithm 2

Figure 3.6 shows the unfolding DAG resulting. Similarly, the probabilities associated with each node are not displayed in order to avoid overloading the DAG. For the sake of illustration, the DAG contains a few nodes that, during the execution of Algorithm 2, are not inserted in either the queue Q or the set D, as their corresponding trace does not belong to the considered log. For example, the left most node in level 6, i.e., $([sink], 6, \langle b, a \rangle)$, resulting from a transition sequence that arrives to $[sink]$ producing the trace $\langle b, a \rangle$, which is not in $\mathcal{T}_{S_{L_2}}$. At level 3, the DAG splits into two parts: the left part corresponds to execution paths producing traces whose first activity is b, whereas the right part corresponds to those whose first activity is a.

Evolution of the queue. We report the contents of the queue Q at early each step of the algorithm, thereby illustrating how the DAG is incrementally constructed, level by level, as the algorithm explores the reachable execution paths. Initially, the queue contains a single item corresponding to the root of the unfolding:

$$Q = \{(([source], 0, \varepsilon), 1)\}$$

The algorithm begins by dequeuing $(([source], 0, \varepsilon), 1)$ and adding its only successor, resulting in:

$$Q = \{(([p_1, p_2], 1, \varepsilon), 1)\}$$

In the marking $[p_1, p_2]$, there are three enabled transitions: a, t_3, and t_4. Thus, dequeuing $(([p_1, p_2], 1, \varepsilon), 1)$ results in adding three new nodes to the queue, yielding:

$$\begin{aligned} Q = \{&(([p_1, p_4], 2, \varepsilon), \frac{w_3}{w_a + w_3 + w_4}), \\ &(([p_2, p_3], 2, \langle a \rangle), \frac{w_a}{w_a + w_3 + w_4}), \\ &(([p_1, p_6], 2, \varepsilon), \frac{w_4}{w_a + w_3 + w_4})\} \end{aligned}$$

Note that, in the example, we express the probabilities as functions of the transition weights. However, during the execution of the algorithm, these computations are performed directly, and only the resulting numerical probabilities are stored. The queue structure ensures that all unfolding nodes at a given level ℓ are explored before any node from the following level $\ell + 1$. This guarantees that we fully enumerate all paths leading to a node and accumulate its exact complete probability mass before proceeding to explore its descendants. From a practical perspective, the order in which elements from the same level are dequeued has no impact on the result and can be chosen arbitrarily, for example, in lexicographic order. We then dequeue the element $(([p_1, p_4], 2, \varepsilon), \frac{w_3}{w_a + w_3 + w_4})$ and append its two descendants to the tail of the queue:

$$\begin{aligned} Q = \{&(([p_2, p_3], 2, \langle a \rangle), \frac{w_a}{w_a + w_3 + w_4}), \\ &(([p_1, p_6], 2, \varepsilon), \frac{w_4}{w_a + w_3 + t_4}), \\ &(([p_1, p_5], 3, \langle b \rangle), \frac{w_3}{w_a + w_3 + t_4} \cdot \frac{w_b}{w_a + w_b}), \\ &(([p_3, p_4], 3, \langle a \rangle), \frac{w_3}{w_a + w_3 + t_4} \cdot \frac{w_a}{w_a + w_b})\} \end{aligned}$$

Then, we dequeue the element $(([p_2, p_3], 2, \langle a \rangle), \frac{w_a}{w_a+w_3+t_4})$, which has three descendants, one of which is already present in the queue. This existing element is currently the last in the queue, with value $(([p_3, p_4], 3, \langle a \rangle), \frac{w_a}{w_a+w_3+t_4} \cdot \frac{w_a}{w_a+w_b})$. Here, the use of the InsertOrIncrease procedure is essential: when a newly discovered execution path ends in a node already present in the queue, we can update its probability by adding the probability of reaching it through this alternative path. After inserting this updated probability and appending the two other descendants, the queue becomes:

$$
\begin{aligned}
Q = \{&(([p_1, p_6], 2, \varepsilon), \frac{w_4}{w_a + w_3 + t_4}),\\
&(([p_1, p_5], 3, \langle b \rangle), \frac{w_3}{w_a + w_3 + t_4} \cdot \frac{w_b}{w_a + w_b}),\\
&(([p_3, p_4], 3, \langle a \rangle), \frac{w_3}{w_a + w_3 + t_4} \cdot \frac{w_a}{w_a + w_b} + \frac{w_a}{w_a + w_3 + t_4} \cdot \frac{w_3}{w_2 + w_3 + w_4}),\\
&(([p_1, p_2], 3, \langle a \rangle), \frac{w_a}{w_a + w_3 + t_4} \cdot \frac{w_2}{w_2 + w_3 + t_4}),\\
&(([p_3, p_6], 3, \langle a \rangle), \frac{w_a}{w_a + w_3 + t_4} \cdot \frac{w_4}{w_2 + w_3 + t_4})\}
\end{aligned}
$$

At each step, only those unfolding nodes that produce a trace belonging to the prefix set of $\mathcal{T}$ are added to the queue. If one of the descendants of a dequeued node reaches the sink marking $[sink]$ and the corresponding trace is in $\mathcal{T}$, then this trace, together with its probability, is inserted into D.

Nature of the probability expressions. In order to illustrate the nature of the expressions generated during the unfolding, and to demonstrate that the probability associated with each node can be computed incrementally from its parent nodes, we present here the equations required to calculate the probability of the sWN trace $\langle b, a \rangle$. For this purpose, we list the probabilities of the nodes involved in the computation of $\mathbb{P}(\langle b, a \rangle)$ as they appear highlighted in the unfolding DAG shown in Figure 3.6. The first two nodes at levels 0 and 1, common to every execution path, are:

$$
\begin{aligned}
&\text{Level 0 :} && \mathbb{P}([source], 0, \varepsilon) = 1\\
&\text{Level 1 :} && \mathbb{P}([p_1, p_2], 1, \varepsilon) = \mathbb{P}([source], 0, \langle\rangle) = 1
\end{aligned}
$$

Then, only one node at level 2 is part of the execution path leading to $\langle b, a \rangle$:

$$
\text{Level 2 for } \langle b, a \rangle : \qquad \mathbb{P}([p_1, p_4], 2, \varepsilon) = \mathbb{P}([p_1, p_2], 1, \varepsilon) \cdot \frac{w_3}{w_a + w_3 + w_4}
$$

We then only have to explore the left branches of the DAG and derive the formula for computing $\mathcal{S}_{S_2}(\langle b, a \rangle, \overline{w})$, which starts with:

$$
\text{Level 3 for } \langle b, a \rangle : \qquad \mathbb{P}([p_1, p_5], 3, \langle b \rangle) = \mathbb{P}([p_1, p_4], 2, \varepsilon) \cdot \frac{w_b}{w_a + w_b}
$$

Then, the execution path splits into two alternatives: firing t_6 first, followed by a, or firing a first, followed by t_6:

$$
\text{Level 4 for } \langle b, a \rangle | \text{ fire } t_6 \text{ first :} \qquad \mathbb{P}([p_1, p_6], 4, \langle b \rangle) = \mathbb{P}([p_1, p_5], 3, \langle b \rangle) \cdot \frac{w_6}{w_a + w_5 + w_6}
$$

Level 4 for $\langle b, a\rangle|$ fire a first : $\mathbb{P}([p_3, p_5], 4, \langle b, a\rangle) = \mathbb{P}([p_1, p_5], 3, \langle b\rangle) \cdot \dfrac{w_a}{w_a + w_5 + w_6}$

Then, the two branches converge at level 5, having fired a first and t_6 second, or vice versa:

$$\text{Level 5 for } \langle b, a\rangle : \quad \mathbb{P}([p_3, p_6], 5, \langle b, a\rangle) = \mathbb{P}([p_1, p_6], 4, \langle b\rangle) \cdot \frac{w_a}{w_a} + \mathbb{P}([p_3, p_5], 4, \langle b, a\rangle) \cdot \frac{w_6}{w_2 + w_5 + w_6}$$

Since this node can be reached through two distinct execution paths, its probability is given by the sum of the probabilities of both paths. From this point, only transition t_7 remains to be fired in order to reach $[sink]$ at level 6:

$$\text{Level 6 for } \langle b, a\rangle : \quad \mathbb{P}([sink], 6, \langle b, a\rangle) = \mathbb{P}([p_3, p_6], 5, \langle b, a\rangle) \cdot \frac{w_7}{w_2 + w_7}$$

In the end, after expansion and simplification, this yields:

$$\mathcal{S}_{S_2}(\langle b, a\rangle, \overline{w}) = \mathbb{P}([sink], 6, \langle b, a\rangle) =$$

$$\frac{w_2 w_4 w_7 (w_a + w_2 + w_3 + w_4)((w_3 + w_4)(w_2 + w_3 + w_4) + w_a(2w_2 + w_3 + w_4 + w_7))}{(w_2 + w_7)^2 (w_2 + w_3 + w_4)^2 (w_a + w_3 + w_4)^2} \tag{3.1}$$

which depends on the value of every weight in the net, except for t_1, which is never in competition with another transition. The influence of each weight, and the relationships between them, are naturally difficult to analyse directly from the formula.

Note that constructing and storing such a formula for every trace that the net can produce quickly becomes infeasible for real-life processes, as both the number of traces and their overall length can explode, resulting in formulas that are impossible to manipulate, even on modern computing systems. The log-driven unfolding DAG enables the compact and efficient storage of all relevant information, structured in a way that allows for the derivation of the restricted stochastic language of the sWN directly. Thus, the procedure can be significantly improved in terms of computation time by incorporating additional components, such as function generation to accelerate the iterative computation of probabilities, and memoisation techniques to avoid redundant recalculations within the code.

On the RG traversal strategy. As stated before, we opted for a breadth-first unfolding of the RG. This scheme guarantees that we systematically discover every transition sequence that produces a given trace. However, other schemes can also be used (e.g., depth-first unfolding, or schemes that take into account the weights to unfold the traces with larger probabilities first). The choice may significantly impact computational time, depending on the sWN and on the set of considered traces.

3.3 Unfolding with dynamic function generation and memoisation

In many applications, the computation of the language emitted by an sWN must be repeated for many different weight vectors $\overline{w}$. This is typically the case within the

realm of discovering the optimal weight vector so that the language of the resulting sWN instance resembles that of the considered log as much as possible.

We have seen that the computation of the language of an sWN with a given weight vector can be achieved via unfolding of the underlying RG; however, changing the weights does not alter the structure of the sWN, meaning that its reachability graph remains the same. As a result, if Algorithm 2 is applied repeatedly to the same sWN, the same unfolding DAG is reconstructed at each iteration, with only the numerical probabilities attached to the nodes changing. Although these probability values differ between iterations, the unfolding computations share a large number of common sub-computations. This redundancy results in considerable inefficiencies in the overall process.

3.3.1 Function generation

To avoid such repetitive computations, we developed an enhanced version of the unfolding procedure in which the unfolding dynamically generates reusable functions during the computation of the stochastic language. This version operates in the same way as Algorithm 2: it follows a breadth-first unfolding scheme and explores the same underlying DAG structure. The key difference lies in the handling of probabilities: instead of propagating numerical values from predecessors to compute each node's probability, this version constructs a function for each node. Each such function computes the node's probability by recursively invoking the functions of its predecessors. The functions take the weight vector as input and return the probability of the corresponding node. As in the queue Q of Algorithm 2, functions are uniquely identified by a triple $(state, level, trace)$, representing the marking, the path length, and the trace prefix. When an equivalent function already exists, its definition is updated to incorporate the new computation path, analogous to the InsertOrIncrease operation in Algorithm 2.

In essence, for a given sWN, a single unfolding suffices to generate a set of functions that compactly encode the probabilistic behaviour of the model. Once constructed, these functions can be directly reused to compute the stochastic language of the sWN induced by any weight vector.

When the computation of the stochastic language is restricted to a subset of the traces that the sWN can generate, such as those observed in the log, this approach offers a substantial advantage: only the nodes indispensable for evaluating these traces are computed. In particular, functions are generated for all nodes during the unfolding, but only those lying on a path that leads to a relevant trace (i.e., a trace present in the log) will ever be invoked. In the DAG depicted in Figure 3.6, if we focus solely on the probabilities of traces represented by the blue leaves (those belonging to the log), the functions associated with nodes that cannot reach any such leaf will never be evaluated. In that example, roughly one quarter of the nodes fall into this category. While this pruning effect can significantly reduce computational cost, its actual impact is difficult to predict in general, as it depends intricately on both the structure of the log and the behaviour of the discovery algorithm employed.

Example 3.1 (Generation of probability function via DAG unfolding).

Considering the unfolding represented by the DAG shown in Figure 3.6, the first function created is associated with the root:

$$\mathcal{F}_{[source],0,\varepsilon}(\overline{w}) := 1$$

and returns 1. Its only successor is associated with the function:

$$\mathcal{F}_{[p_1,p_2],1,\varepsilon}(\overline{w}) := 1 \cdot \mathcal{F}_{[source],0,\varepsilon}(\overline{w})$$

By considering the successors of node $([p_2, p_4], 1, \varepsilon)$, the functions corresponding to nodes of level 2 are created as:

$$\begin{aligned}
\mathcal{F}_{[p_1,p_4],2,\varepsilon}(\overline{w}) &:= \frac{w_3}{w_a + w_3 + w_4} \cdot \mathcal{F}_{[p_1,p_2],1,\varepsilon}(\overline{w}) \\
\mathcal{F}_{[p_2,p_3],2,\langle a\rangle}(\overline{w}) &:= \frac{w_a}{w_a + w_3 + w_4} \cdot \mathcal{F}_{[p_1,p_2],1,\varepsilon}(\overline{w}) \\
\mathcal{F}_{[p_1,p_6],2,\varepsilon}(\overline{w}) &:= \frac{w_4}{w_a + w_3 + w_4} \cdot \mathcal{F}_{[p_1,p_2],1,\varepsilon}(\overline{w})
\end{aligned}$$

The successors of node $([p_2, p_5], 2, \varepsilon)$ yields the creation of functions:

$$\begin{aligned}
\mathcal{F}_{[p_1,p_5],3,\langle b\rangle}(\overline{w}) &:= \frac{w_b}{w_a + w_b} \cdot \mathcal{F}_{[p_1,p_4],2,\varepsilon}(\overline{w}) \\
\mathcal{F}_{[p_3,p_4],3,\langle a\rangle}(\overline{w}) &:= \frac{w_a}{w_a + w_b} \cdot \mathcal{F}_{[p_1,p_4],2,\varepsilon}(\overline{w})
\end{aligned}$$

When considering the three successors of node $([p_2, p_3], 2, \langle a\rangle)$, the function $\mathcal{F}_{[p_3,p_4],3,\langle a\rangle}(\overline{w})$ must be updated because it already exists and two new ones are created. This results in:

$$\begin{aligned}
\mathcal{F}_{[p_3,p_4],3,\langle a\rangle}(\overline{w}) &:= \frac{w_a}{w_a + w_b} \cdot \mathcal{F}_{[p_1,p_4],2,\varepsilon}(\overline{w}) + \frac{w_3}{w_2 + w_3 + w_4} \cdot \mathcal{F}_{[p_2,p_3],2,\langle a\rangle}(\overline{w}) \\
\mathcal{F}_{[p_1,p_2],3,\langle a\rangle}(\overline{w}) &:= \frac{w_2}{w_2 + w_3 + w_4} \cdot \mathcal{F}_{[p_2,p_3],2,\langle a\rangle}(\overline{w}) \\
\mathcal{F}_{[p_3,p_6],3,\langle a\rangle}(\overline{w}) &:= \frac{w_4}{w_2 + w_3 + w_4} \cdot \mathcal{F}_{[p_2,p_3],2,\langle a\rangle}(\overline{w})
\end{aligned}$$

From a practical standpoint, we create, for each node, a function that takes a weight vector as a variable input and, at creation time, is also provided with fixed additional information: the set of transitions enabled in the corresponding marking and the specific transition to be fired to match the behaviour of the node. To achieve this, we utilise partial functions, which enable us to pre-bind these fixed parameters at function creation. This way, when the function is later called with a different weight vector, it can directly compute the probability of the node based on the fixed information.

3.3.2 Memoisation

A further important optimisation leverages the structural properties of the unfolding DAG. While in a tree, a node has exactly one parent, and thus can only appear along a single unique path from the root to any given leaf, in a DAG, such as the one obtained by unfolding the reachability graph of an sWN, a node may lie on multiple distinct paths from the root to the same leaf. Consequently, when computing the probability of a given trace (i.e., reaching a leaf in the sink marking), the probability of specific intermediate nodes may be needed multiple times, as all contributing paths must be considered. This redundancy becomes even more significant when computing the probabilities of many traces, as is required when evaluating the whole stochastic language. To eliminate repeated calculations, we employ a memoisation mechanism: a hash table stores the probability of each node immediately after it is computed, allowing subsequent calls to reuse the stored value instead of re-evaluating the corresponding recursive computation.

To sum up, combining the unfolding procedure, which generates reusable functions that capture the probabilistic behaviour of the net, with memoisation during evaluation significantly improves computational efficiency. This synergy substantially reduces the overall runtime. This version of the algorithm will be referred to in the remainder of the thesis as the "memoised unfolding". To use this version, a preliminary setup must be performed to build the unfolding DAG and generate the symbolic functions necessary for probability computations. This setup function is denoted as $\textsc{UnfoldSetup}(\mathcal{T},N,R_g(N))$, where $\mathcal{T}$ is the set of traces to retrieve in the model N, based on its reachability graph $R_g(N)$. It is important to note that this algorithm is applied directly to a non-stochastic workflow net, since the weight vector is not yet relevant at this stage.

3.4 Silent loops in sWN language computation

It is worth noting an important limitation that concerns the reachability graph unfolding-based procedure for computing an sWN stochastic language, introduced in Section 3.2. In fact, a necessary condition for such a procedure to correctly operate is the absence of silent loops (i.e., loops of silent transitions) in the considered sWN model. If silent loops are present, then infinitely many paths in the reachability graph exist that yield the same trace of the log. Therefore, the computation of the sWN stochastic language would become an unfeasible task.

This special situation was encountered in the BPI Challenge 2012 [77], which describes a loan application process. Figure 3.7 shows a small portion of the workflow net discover with Inductive Miner that contains a loop that can generate transition paths that consist solely of repeated silent transitions. After firing t_1, the resulting conflict may lead to firing t_2, after which the model can loop back by firing t_6. This behaviour creates, within the net, an infinite number of distinct transition paths that all generate the same trace in the net language.

Extending this phenomenon to the stochastic setting implies that computing the probability of certain traces becomes impossible when using unfolding-based strategies. Since the unfolding method relies on enumerating all transition sequences that generate a given trace, and this number becomes infinite in the presence of silent loops, the procedure inevitably enters an infinite loop while attempting to construct an ever-growing DAG. A possible solution to this issue is to detect such patterns and then analytically exploit the geometric behaviour induced by loops in sWNs. In particular, when a loop consists solely of silent transitions, the probability mass associated with the infinitely many corresponding paths can often be expressed in closed form using geometric-series arguments. Alternatively, one may remove these loops by simplifying the net while preserving the same language.

While silent loops are a clear conceptual limitation of the unfolding procedure, it is less clear how significant their impact is in practice. Although the Inductive Miner, in its original formulation, can in principle produce workflow nets containing silent loops, it is not straightforward to characterise the specific conditions under which an event log would lead to such a structure. Moreover, certain implementations of the Inductive Miner apply language-preserving reduction rules [60, 38]. These rules (e.g., place merging or transition elimination) can reduce the structure of the discovered workflow net and, in some cases, eliminate simple forms of silent loops.

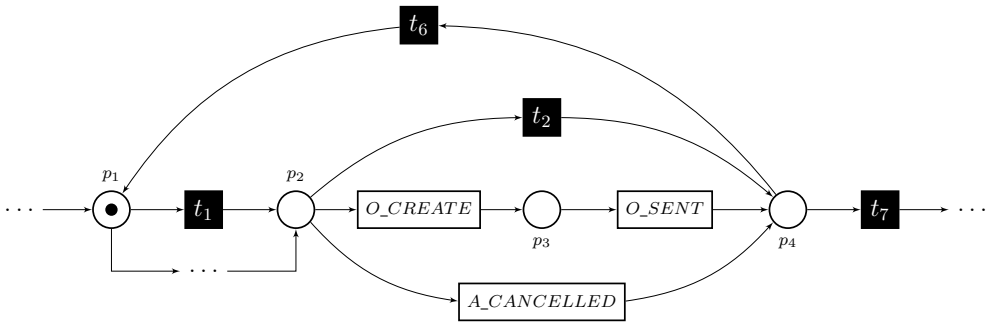


Figure 3.7 – Excerpt of the WN discovered by the Inductive Miner from the BPI Challenge 2012 log, highlighting the presence of a silent loop

3.5 Conclusion

In this chapter, we have introduced a log-driven breadth-first unfolding procedure for computing the restricted stochastic language of an sWN. This procedure relies on constructing a DAG, which compactly captures all execution paths relevant to a given set of traces, typically those observed in the event log. By propagating probabilities along the unfolding structure, we obtain the probability of each trace in the log, thereby enabling the evaluation of stochastic distance measures between the model and the log.

We have further proposed an enhanced version of the unfolding, referred to as the memoised unfolding, which replaces direct numerical computation with the generation of reusable functions encoding the probabilistic behaviour of each un-

folding node. Combined with memoisation, this approach avoids redundant recomputations when evaluating the stochastic language under multiple weight vectors, yielding substantial efficiency gains, particularly in iterative contexts such as optimisation.

Overall, the unfolding approach presented here provides a flexible and efficient foundation for the stochastic process discovery framework developed in this thesis. In the next chapter, we build upon this foundation to define an optimisation procedure, based on the log-driven unfolding method, that aims to discover an sWN as close as possible to a given log according to predefined distance metrics.

Chapter 4

Stochastic Process Discovery via Optimisation

Many practical and sophisticated algorithms have been proposed to derive a control-flow model that accurately reproduces the language of a given event log [71, 86, 49]. However, when the goal shifts from reproducing the structure of the behaviour to capturing its stochastic characteristics, far fewer methods have been developed, despite a growing interest in recent years. Now that we can rely on an effective procedure to compute the exact restricted stochastic language of a stochastic workflow net (sWN) (see Chapter 3), we can integrate this into a framework for indirectly discovering and optimising an sWN whose stochastic behaviour matches, as closely as possible, that observed in a given event log. This framework belongs to the family of parameter estimation techniques and, more specifically, to optimisation-based approaches. In this setting, the model's structure is assumed to be fixed, obtained using a standard control-flow discovery algorithm. At the same time, its stochastic parameters are iteratively adjusted to minimise a quantitative measure of dissimilarity between the model and the observed log behaviour.

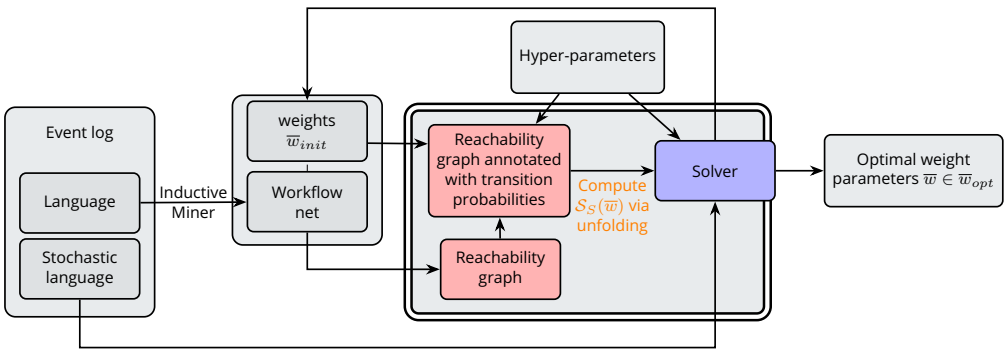


Figure 4.1 – Optimised stochastic process discovery framework

As illustrated in Figure 4.1, the approach begins with an event log. A workflow net (WN) is first discovered using a standard control-flow process discovery algorithm. This net is then transformed into an sWN by augmenting its definition with a weight assignment function W. The weight function W is initialised according to a vector $\overline{w}_{init}$ which assigns a weight to every transition of the model. This initial weight vector is selected through a simple random uniform sampling of the weight space, and its construction will be discussed in detail later. This initial sWN, together with the stochastic language of the log that it must match, and a set of hyperparameters configuring the unfolding and optimisation components of the framework, are provided as input to the main procedure. At the end of the process, the optimiser is expected to produce an optimal sWN whose stochastic language matches, as closely as possible, the stochastic language of the event log. In practice, the output consists of a new optimised weight vector $\overline{w}_{opt}$ that can be associated with the initial WN to construct the optimised sWN. This entire process enables the inference of a probabilistically accurate model from observed data in a fully automated and data-driven manner. The remainder of this chapter is structured as follows:

- Section 4.1 outlines two quantitative distance functions we used to assess the stochastic conformance between the log's and the sWN languages: the Kullback-Leibler divergence (KLD) and the Earth Mover's Distance (EMD). We present their mathematical formulations and discuss their relevance in the context of stochastic model fitting.

- Section 4.2 analyses the analytical properties of the distance functions, particularly focusing on differentiability and structural characteristics. We illustrate how these properties influence optimisation strategies.

- Section 4.3 introduces the framework and describes the optimisation scheme to infer transition weights from an event log and a mined model. We also detail the solvers employed (gradient-based and derivative-free).

- Section 4.4 investigates the factors affecting the scalability and computational cost of our approach. We examine the impact of the size and structure of the event log, the mined model, and the unfolding DAG, and compare the trade-offs between memoised and non-memoised strategies.

- Section 4.5 presents an extensive experimental evaluation. We analyse the framework's performance on several real-life event logs, compare the effectiveness of different solvers and unfolding configurations, and evaluate convergence behaviour. We also compare our method against state-of-the-art indirect stochastic process discovery approaches.

- Section 4.6 concludes the chapter and points to Hessian-based optimisation as a possible and easily extendable direction.

4.1 Measuring the distance between two stochastic languages

The core problem addressed in this thesis can be stated as follows: given a stochastic model mined from an event log, the objective is to determine a stochastic parameter vector such that the resulting stochastic language closely matches the empirical stochastic language derived from the log. In this chapter, we consider an sWN S, parameterised by a weight vector $\overline{w}$, and define distance measures to quantify the similarity between its stochastic language and the empirical stochastic language induced by an event log. To this end, a crucial step is to define a meaningful and computable measure of difference between the two stochastic languages. The choice of this distance function directly impacts the optimisation process, as it determines the shape of the objective function and the type of solution that is favoured. In this section, we adapt our setting and consider two distance-based formulations: a Maximum Likelihood Estimation approach, equivalently expressed as the minimisation of the Kullback–Leibler divergence (KLD), and an alternative measure grounded in optimal transport, the Earth Mover's Distance, restricted to the log support. Each of them captures different aspects of the mismatch between the two distributions and presents distinct algorithmic and numerical properties.

Note that we evaluate the difference between logs and models in terms of a *distance*, as opposed to most of the process mining literature, which assesses model quality based on a *level of conformance*. The key distinction is that, under a distance-based approach, a perfect match between the model and the log yields a value of 0. In contrast, conformance measures typically assign a value of 1 to denote perfect alignment. This distinction is introduced because it is more natural, in our context, to formulate an optimisation problem as one of *minimising a distance* between the model and the log. However, this is ultimately a matter of perspective, as framing the problem as one of *maximising a conformance score* would lead to the same result from an optimisation standpoint. We point out here that both distance measures defined in this section:

- the log-likelihood fonction, $\log(M_{LH}(\overline{w}))$, given in Equation (4.2), and
- the rEMD function, $M_{rEMD}(\overline{w})$, given in Equation (4.4),

can be computed directly from the trace probabilities returned by $\textsc{UnfoldRG}(\mathcal{T}_L, S, R_g(S))$ (Algorithm 2). Alternatively, the more efficient memoised variant described in Section 3.3 can be employed, which is particularly advantageous when these computations are performed repeatedly within an optimisation loop

4.1.1 Maximum likelihood estimation

One commonly used distance measure is the KLD. Its use aligns with what is arguably the most widespread approach to parameter estimation: maximum likelihood estimation. In this context, the goal is to find the point in the parameter space

(i.e., the vector of transition weights $\overline{w}$) for which the observed log L is most likely under the model S. The corresponding optimisation procedure is built upon the likelihood function, which evaluates, for a given weight vector, the probability that the sWN generates the observed event log. Maximising this likelihood is therefore equivalent to minimising the KLD between the empirical distribution $\mathcal{S}_L$ and the model distribution $\mathcal{S}_S$. The likelihood function between $\mathcal{S}_L$ and $\mathcal{S}_S$ is given by:

$$M_{LH}(\overline{w}) = \prod_{\sigma \in \mathcal{T}_L} \mathcal{S}_S(\sigma, \overline{w})^{\mathcal{S}_L(\sigma)}. \tag{4.1}$$

Since Equation (4.1) is prone to numerical underflow, especially when dealing with long traces or small probabilities, it is common, for optimisation purposes, to work with its logarithmic form. The logarithm preserves the location of the maximum and transforms the product of probabilities into a sum of log-probabilities, which is numerically more stable:

$$\log(M_{LH}(\overline{w})) = \sum_{\sigma \in \mathcal{T}_L} \mathcal{S}_L(\sigma) \cdot \log(\mathcal{S}_S(t, \overline{w})) \tag{4.2}$$

which is called the log-likelihood function. The use of the logarithm is justified by the fact that we are dealing exclusively with strictly positive probability values. It is important to note that the computation of Equation (4.2) requires the probabilities of only those traces that are actually present in the event log L. This aligns perfectly with the unfolding procedure described earlier, which is log-driven and restricts its exploration to the traces observed in the log.

4.1.2 Restricted Earth Mover's Distance

Another widely used method for quantifying the difference between two distributions is the Earth Mover's Distance (EMD), also known as the Wasserstein distance. To define the EMD in our context, we adapt the formulation of the Earth Mover's Stochastic Conformance (EMSC) measure introduced by Leemans et al. [52]. The Earth Mover's Distance (EMD) between the empirical distribution $\mathcal{S}_L$ (derived from the event log) and the model distribution $\mathcal{S}_S$, which depends on the weight vector $\overline{w}$, is defined through the following optimisation problem. This formulation captures the minimal cost required to transform one distribution into the other:

$$M_{EMD}(\overline{w}) = \min_{r \in \mathcal{R}} \sum_{\sigma_1 \in \mathcal{T}_L} \sum_{\sigma_2 \in \mathcal{T}_S} r(\sigma_1, \sigma_2)\, d(\sigma_1, \sigma_2) \tag{4.3}$$

$$\text{such that } \begin{cases} \sum_{\sigma_2 \in \mathcal{T}_S} r(\sigma_1, \sigma_2) = \mathcal{S}_L(\sigma_1) & \forall \sigma_1 \in \mathcal{T}_L \\ \sum_{\sigma_1 \in \mathcal{T}_L} r(\sigma_1, \sigma_2) = \mathcal{S}_S(\sigma_2, \overline{w}) & \forall \sigma_2 \in \mathcal{T}_S \\ r(\sigma_1, \sigma_2) \geq 0 & \forall \sigma_1 \in \mathcal{T}_1, \sigma_2 \in \mathcal{T}_2 \end{cases}$$

where the function $r(\sigma_1, \sigma_2)$ (with $\sigma_1 \in \mathcal{T}_L$ and $\sigma_2 \in \mathcal{T}_S$) is the reallocation function.

In Equation (4.3), computing the EMD requires finding a reallocation function r that transforms the stochastic language $\mathcal{S}_L$ (derived from the log L) into the stochastic language $\mathcal{S}_S$ (defined by the sWN S with weights $\overline{w}$) with minimal total cost in the infinite set of valid reallocation function $\mathcal{R}$. This cost corresponds to the amount of probability mass moved, weighted by the distance $d(\sigma_1, \sigma_2)$ between traces $\sigma_1 \in \mathcal{T}_L$ and $\sigma_2 \in \mathcal{T}_S$. The first two constraints on r ensure that it behaves as a valid transport plan: it fully redistributes the probability mass of $\mathcal{S}_L$ and exactly reconstructs $\mathcal{S}_S$. The third constraint excludes moving negative probability masses.

In practice, the optimal redistribution matrix is obtained by solving a standard optimisation problem, for which efficient algorithms are available. Depending on the size of the trace sets and the desired trade-off between accuracy and computational cost, this optimisation can be solved exactly using linear programming, or approximated using faster methods that scale better to large supports. In particular, approximate approaches rely on problem relaxations or regularisation terms, enabling the computation of a near-optimal redistribution.

Using the normalised version of the Levenshtein distance, it is guaranteed that $0 \leq M_{EMD}(\overline{w}) \leq 1$, resulting in a measure that, conversely to the log-likelihood function, is possible to interpret (i.e., 0 indicates perfect matching while 1 complete difference). If $\mathcal{T}_L$ and $\mathcal{T}_S$ are finite sets, computing (4.3) corresponds to a linear programming problem whose size depends on the union $|\mathcal{T}_L \cup \mathcal{T}_S|$.

When the model set of traces $\mathcal{T}_S$ is infinite, as is often the case, since process discovery algorithms frequently produce nets with loops, the exact computation of the EMD becomes infeasible. In such situations, it can be approximated by the so-called truncated EMD (tEMD) [52]. The core idea behind tEMD is to consider a sufficiently large finite subset of traces from $\mathcal{T}_S$ that collectively account for a specified fraction of the total probability mass. However, in practice, even moderate coverage thresholds (e.g., 0.8) may require the unfolding of a large number of traces, significantly increasing the computational cost. As a result, tEMD is often too slow to be used effectively within an iterative parameter estimation procedure.

As an alternative and to fit the log-driven unfolding, we propose an alternative approach, which we name restricted EMD (rEMD), that is, to restrict the calculation of the EMD to those traces that are present in the event log. Note that the EMD defined in (4.3) can only be computed between two proper probability distributions. However, restricting the stochastic language of the model to the traces observed in the log generally produces an incomplete distribution, whose total probability mass is strictly less than one. To compute the rEMD correctly under these conditions, we normalise the stochastic language of the net by dividing each probability value by the total mass of the restricted language. More precisely, once the restricted stochastic language $\mathcal{S}'_S$ of the sWN is extracted, whether from the unfolding or by any other technique, we compute its total probability mass $Z_{S'}$. Each probability assigned to a trace $\sigma \in \mathcal{T}_S \cap \mathcal{T}_L$ is then normalised:

$$M_{rEMD}(\overline{w}) = \min_{r \in \mathcal{R}'} \sum_{\sigma_1 \in \mathcal{T}_L} \sum_{\sigma_2 \in \mathcal{T}_S \cap \mathcal{T}_L} r(\sigma_1, \sigma_2)\, d(\sigma_1, \sigma_2) \tag{4.4}$$

$$
\text{such that} \begin{cases} Z_{S'} = \sum\limits_{\sigma \in \mathcal{T}_S \cap \mathcal{T}_L} \mathcal{S}'_S(\sigma, \overline{w}) & \\ \mathcal{S}_S(\sigma, \overline{w}) = \dfrac{\mathcal{S}'_S(\sigma, \overline{w})}{Z_{S'}} & \forall \sigma \in \mathcal{T}_S \cap \mathcal{T}_L \\ \sum\limits_{\sigma_2 \in \mathcal{T}_S \cap \mathcal{T}_{\mathcal{S}_L}} r(\sigma_1, \sigma_2) = \mathcal{S}_L(\sigma_1) & \forall \sigma_1 \in \mathcal{T}_L \\ \sum\limits_{\sigma_1 \in \mathcal{T}_L} r(\sigma_1, \sigma_2) = \mathcal{S}_S(\sigma_2, \overline{w}) & \forall \sigma_2 \in \mathcal{T}_S \cap \mathcal{T}_L \\ r(\sigma_1, \sigma_2) \geq 0 & \forall \sigma_1, \sigma_2 \text{ in domain} \end{cases}
$$

In most cases, $|\mathcal{T}_L| \ll |\mathcal{T}_S|$ even when $\mathcal{T}_S$ is not infinite. Hence reducing the calculation of the EMD to the traces in $\mathcal{T}_L$ can be feasible to optimisation purposes.

4.2 Characteristics of the objective function and its partial derivatives

Gradient-based optimisation methods require the computation of the partial derivatives of the objective function with respect to the variables being optimised. When analytical expressions for these derivatives are not available, they can be approximated numerically using finite difference methods. Although numerical approximation is often sufficient, closed-form derivatives are generally preferable, offering higher precision and, in many cases, improved computational efficiency.

In Section 4.2.1, we derive the partial derivatives of the objective function in the case where the log-likelihood function is used (Equation (4.2)), thereby enabling the use of exact gradients during optimisation. In contrast, when using the rEMD, the dependency of the objective function on the model parameters is significantly more intricate (Equation (4.4)). Consequently, the derivatives with respect to the weights cannot be expressed in closed form and must be approximated numerically. In Section 4.2.2, we investigate the behaviour of the rEMD-based objective function through a simple example, which serves to highlight the inherent challenges and limitations of employing rEMD in gradient-based optimisation.

4.2.1 Partial derivatives in case of using the log-likelihood function

To derive the partial derivative of log-likelihood function with respect to the model parameters, we assume that the sWN contains n transitions, and denote the weight vector as $\overline{w} = (w_1, w_2, \ldots, w_n)$, where each weight is indexed by a natural number. Under this notation, the partial derivative of $log(M_{LH})$ (Equation (4.2)) with respect to a specific weight w_i, for $1 \leq i \leq n$, can be expressed as:

$$
\frac{\partial \log(M_{LH}(\overline{w}))}{\partial w_i} = \sum_{\sigma \in \mathcal{T}_{\mathcal{S}_L}} \mathcal{S}_L(\sigma) \frac{1}{\mathcal{S}_S(\sigma, \overline{w})} \cdot \frac{\partial \mathcal{S}_S(\sigma, \overline{w})}{\partial w_i}
$$

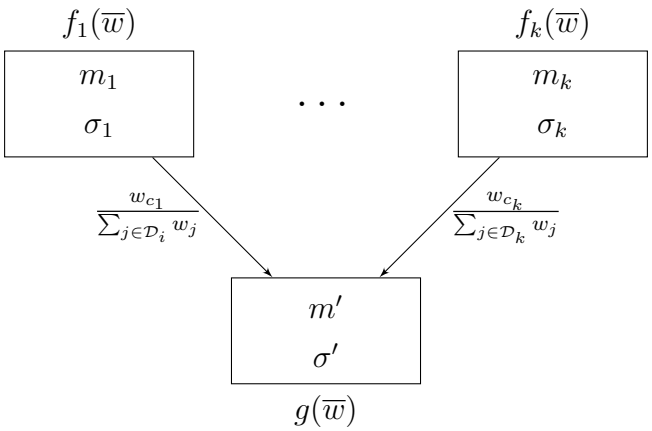


Figure 4.2 – A generic node of the unfolding with its predecessors

The partial derivative $\frac{\partial \mathcal{S}_S(\sigma,\overline{w})}{\partial w_i}$ can be computed by extending the unfolding procedure so that, for each node, we compute not only its probability but also its partial derivatives with respect to all weights $w_1, w_2, \ldots, w_n$. As a result, each node in the unfolding will be associated with $n+1$ numerical values: its probability, and one partial derivative per weight. This enriched unfolding enables the gradient of the log-likelihood function to be computed efficiently in a single pass over the unfolding structure.

The probability associated with the root of the unfolding is 1 and, since its value does not depend on the weights, its partial derivatives with respect to the weights are equal to 0. We now address the problem of determining the partial derivatives of a node's probability based on the partial derivatives of its predecessor nodes. To this end, consider a generic node whose probability, as a function of the weight vector $\overline{w}$, is denoted by $g(\overline{w})$, and suppose that it has k predecessors. Let the probabilities of these k predecessors be denoted by $f_i(\overline{w})$, for $1 \leq i \leq k$, and assume that the partial derivatives of these probabilities with respect to the weights are known. Furthermore, let w_{c_i} be the weight of the transition whose firing leads from the i-th predecessor to the current node and let $\mathcal{D}_i$ denote the set of indices corresponding to the weights of the transitions enabled at the i-th predecessor. These weights collectively form the denominator in the computation of the firing probability along each incoming arc in the unfolding. With this notation, Figure 4.2 illustrates the relationship between the function g and its k predecessor functions f_i. Specifically, the probability $g(\overline{w})$ associated with the node can be expressed as:

$$g(\overline{w}) = \sum_{i=1}^{k} f_i(\overline{w}) \frac{w_{c_i}}{\sum_{j\in\mathcal{D}_i} w_j}$$

The partial derivative of $g(\overline{w})$ with respect to a weight w_l, for $1 \leq l \leq n$, can be expressed in terms of the probabilities and partial derivatives of the predecessor nodes as follows:

$$\frac{\partial g(\overline{w})}{\partial w_l} = \sum_{i=1}^{k} \left(\frac{\partial f_i(\overline{w})}{\partial w_l} \cdot \frac{w_{c_i}}{\sum_{j\in\mathcal{D}_i} w_j} + f_i(\overline{w}) \frac{\partial \frac{w_{c_i}}{\sum_{j\in\mathcal{D}_i} w_j}}{\partial w_l} \right) \tag{4.5}$$

where

$$\frac{\partial \frac{w_{c_i}}{\sum_{j\in\mathcal{D}_i} w_j}}{\partial w_l} = \begin{cases} 0 & \text{if } l \notin \mathcal{D}_i \\ -\dfrac{w_{c_i}}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^2} & \text{if } l \in \mathcal{D}_i \wedge l \neq c_i \\ \dfrac{\sum_{j\in\mathcal{D}_i \wedge j\neq l} w_j}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^2} & \text{if } l \in \mathcal{D}_i \wedge l = c_i \end{cases} \tag{4.6}$$

where note that $c_i \in \mathcal{D}_i$ always holds.

To illustrate the procedure, we revisit the sWN S_2 shown in Figure 3.4 together with its corresponding unfolding DAG, depicted in Figure 3.6. We analyse how the probability of generating the sequence $\langle b, a\rangle$, denoted by $\mathcal{S}_{S_2}(\langle b, a\rangle, \overline{w})$ and given in Equation (3.1), depends on the weight w_5.

As explained earlier, the probability of the initial unfolding node $([source], 0, \varepsilon)$ is always equal to 1 and does not depend on $\overline{w}$. Therefore, its partial derivative with respect to w_5 is 0, as is the case for any other weight:

$$\frac{\partial \mathbb{P}([source], 0, \varepsilon)}{\partial w_5} = 0$$

The following three node that are on the execution path to produce $\langle b, a\rangle$, their probability doesn't involved w_5 as is it now enabled in their corresponding markings:

$$\frac{\partial \mathbb{P}([p_1, p_2], 1, \varepsilon)}{\partial w_5} = 0 \qquad \frac{\partial \mathbb{P}([p_1, p_4], 2, \varepsilon)}{\partial w_5} = 0 \qquad \frac{\partial \mathbb{P}([p_1, p_5], 3, \langle b\rangle)}{\partial w_5} = 0$$

We then reach the unfolding node $([p_1, p_5], 3, \langle b\rangle)$. Following the reachability graph $R_g(S_2)$ in Figure 3.5, we observe that w_5 is enabled in marking $[p_1, p_5]$. To compute the probability of reaching the unfolding node $([sink], 6, \langle b, a\rangle)$, which determines $\mathcal{S}_{S_2}(\langle b, a\rangle, \overline{w})$, we must consider the two possible cases:

1. Firing transition t_6 in marking $[p_1, p_5]$, whose partial derivative with respect to w_5 is:

$$\begin{aligned} \frac{\partial \mathbb{P}([p_1, p_6], 4, \langle b\rangle)}{\partial w_5} &= \frac{\partial \mathbb{P}([p_1, p_5], 3, \langle b\rangle)}{\partial w_5} \cdot \frac{w_6}{w_a + w_5 + w_6} \\ &\quad + \mathbb{P}([p_1, p_5], 3, \langle b\rangle) \cdot \frac{\partial}{\partial w_5}\left(\frac{w_6}{w_a + w_5 + w_6}\right) \\ &= -\mathbb{P}([p_1, p_5], 3, \langle b\rangle) \cdot \frac{w_6}{(w_a + w_5 + w_6)^2}. \end{aligned}$$

2. Firing transition a in the same marking, whose partial derivative with respect to w_5 is:

$$\frac{\partial \mathbb{P}([p_3, p_5], 4, \langle b, a\rangle)}{\partial w_5} = \frac{\partial \mathbb{P}([p_1, p_5], 3, \langle b\rangle)}{\partial w_5} \cdot \frac{w_a}{w_a + w_5 + w_6}$$

$$
\begin{aligned}
&+ \mathbb{P}([p_1, p_5], 3, \langle b \rangle) \cdot \frac{\partial}{\partial w_5} \left(\frac{w_a}{w_a + w_5 + w_6} \right) \\
&= -\mathbb{P}([p_1, p_5], 3, \langle b \rangle) \cdot \frac{w_a}{(w_a + w_5 + w_6)^2}.
\end{aligned}
$$

Then, both nodes $([p_1, p_6], 4, \langle b \rangle)$ and $([p_3, p_5], 4, \langle b, a \rangle)$ share a common descendant in the DAG, which lies on the execution path producing $\langle b, a \rangle$:

$$
\begin{aligned}
\frac{\partial \mathbb{P}([p_3, p_6], 5, \langle b, a \rangle)}{\partial w_5} &= \frac{\partial \mathbb{P}([p_1, p_6], 4, \langle b \rangle)}{\partial w_5} \cdot \frac{w_a}{w_a} + \mathbb{P}([p_1, p_6], 4, \langle b \rangle) \cdot \frac{\partial}{\partial w_5} \left(\frac{w_a}{w_a} \right) \\
&\quad + \frac{\partial \mathbb{P}([p_3, p_5], 4, \langle b, a \rangle)}{\partial w_5} \cdot \frac{w_6}{w_2 + w_5 + w_6} \\
&\quad + \mathbb{P}([p_3, p_5], 4, \langle b, a \rangle) \cdot \frac{\partial}{\partial w_5} \left(\frac{w_6}{w_2 + w_5 + w_6} \right) \\
&= \frac{\partial \mathbb{P}([p_1, p_6], 4, \langle b \rangle)}{\partial w_5} + \frac{\partial \mathbb{P}([p_3, p_5], 4, \langle b, a \rangle)}{\partial w_5} \cdot \frac{w_6}{w_2 + w_5 + w_6} \\
&\quad - \mathbb{P}([p_3, p_5], 4, \langle b, a \rangle) \cdot \frac{w_6}{(w_2 + w_5 + w_6)^2}.
\end{aligned}
$$

And then we finally reach the final node $([sink], 6, \langle b, a \rangle)$:

$$
\begin{aligned}
\frac{\partial \mathbb{P}([sink], 6, \langle b, a \rangle)}{\partial w_5} = \frac{\partial \mathcal{S}_{S_2}(\langle b, a \rangle, \overline{w})}{\partial w_5} &= \frac{\partial \mathbb{P}([p_3, p_6], 5, \langle b, a \rangle)}{\partial w_5} \cdot \frac{w_7}{w_2 + w_7} \\
&\quad + \mathbb{P}([p_3, p_6], 5, \langle b, a \rangle) \cdot \frac{\partial}{\partial w_5} \left(\frac{w_7}{w_2 + w_7} \right) \\
&= \frac{\partial \mathbb{P}([p_3, p_6], 5, \langle b, a \rangle)}{\partial w_5} \cdot \frac{w_7}{w_2 + w_7}
\end{aligned}
$$

As expected, the derivative associated with a given node is expressed as a function of both the probability and the partial derivatives of its parent nodes. The explicit expression for $\frac{\partial \mathcal{S}_{S_2}(\langle b, a \rangle, \overline{w})}{\partial w_5}$ is

$$
\frac{w_b w_3 w_6 w_7 (w_a (2(w_5 + w_6) + w_2) + (w_2 + w_5 + w_6)^2 + w_a^2)}{(w_2 + w_7)(w_a + w_b)(w_2 + w_5 + w_6)^2 (w_a + w_5 + w_6)^2 (w_a + w_3 + w_4)}
$$

The computation of the first-order partial derivatives can be integrated directly into Algorithm 2 by calculating, at each node, the numerical values of its probability together with the required derivatives, based on those of its parent nodes. However, when derivatives are required repeatedly, for instance, during optimisation over multiple weight vectors, this direct numerical approach can become inefficient. In such cases, it is advantageous to adopt the memoised unfolding strategy described in Section 3.3. With this approach, a single unfolding is performed, during which symbolic expressions (i.e., parameterised functions) for the probabilities and their derivatives are constructed. These functions can then be evaluated for any given parameter vector, and their results reused via memoisation, thereby eliminating redundant computations and improving overall performance.

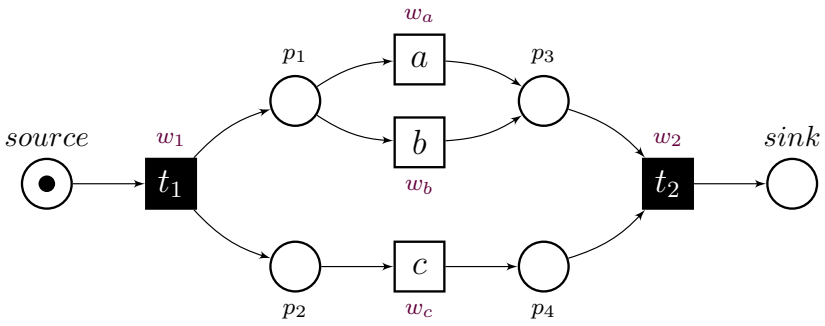


Figure 4.3 – sWN S_3 derived from the log L_3

4.2.2 Objective function and its derivatives in case of using the restricted Earth Mover's Distance

In the following, we illustrate the behaviour of the rEMD as a function of the weights through a simple example. We consider the log L_3 with its four traces with the following probabilities:

$$\mathcal{S}_{L_3}(\langle a, c\rangle) = 0.3 \qquad \mathcal{S}_{L_3}(\langle c, a\rangle) = 0.1$$
$$\mathcal{S}_{L_3}(\langle b, c\rangle) = 0.2 \qquad \mathcal{S}_{L_3}(\langle c, b\rangle) = 0.4$$

together with the sWN S_3 depicted in Figure 4.3 producing the four sequences with probabilities:

$$\mathcal{S}_{S_3}(\langle a, c\rangle) = \frac{w_a}{w_a + w_b + w_c} \qquad \mathcal{S}_{S_3}(\langle c, a\rangle) = \frac{w_c}{w_a + w_b + w_c} \cdot \frac{w_a}{w_a + w_b}$$

$$\mathcal{S}_{S_3}(\langle b, c\rangle) = \frac{w_b}{w_b + w_c + w_d} \qquad \mathcal{S}_{S_3}(\langle c, b\rangle) = \frac{w_c}{w_a + w_b + w_c} \cdot \frac{w_b}{w_a + w_b}$$

The weights of the silent transitions t_1 and t_2 have no impact on the behaviour of the sWN, since they are never in conflict with any other transition.

Figure 4.4 depicts the value of M_{rEMD} and the M_{LH} as a function of w_a, while keeping $w_b = w_c = 1$ fixed. The plot reveals several important characteristics of the rEMD. First, there exists an interval in which the rEMD remains constant, meaning that its partial derivative with respect to w_a is zero. This is particularly noteworthy because, within the same interval, the probabilities assigned to the four considered sequences vary with w_a. This illustrates a fundamental limitation of the rEMD in optimisation contexts: the objective function may exhibit flat regions where changes in the model parameters do not result in any improvement, despite underlying changes in the model's behaviour. The reason of the zero partial derivative is the following. In the interval in question we have

$$\mathcal{S}_{L_3}(\langle a, c\rangle) > \mathcal{S}_{S_3}(\langle a, c\rangle), \qquad \mathcal{S}_{L_3}(\langle c, a\rangle) > \mathcal{S}_{S_3}(\langle c, a\rangle),$$
$$\mathcal{S}_{L_3}(\langle c, a\rangle) < \mathcal{S}_{S_3}(\langle b, c\rangle), \qquad \mathcal{S}_{L_3}(\langle c, b\rangle) > \mathcal{S}_{S_3}(\langle c, b\rangle),$$

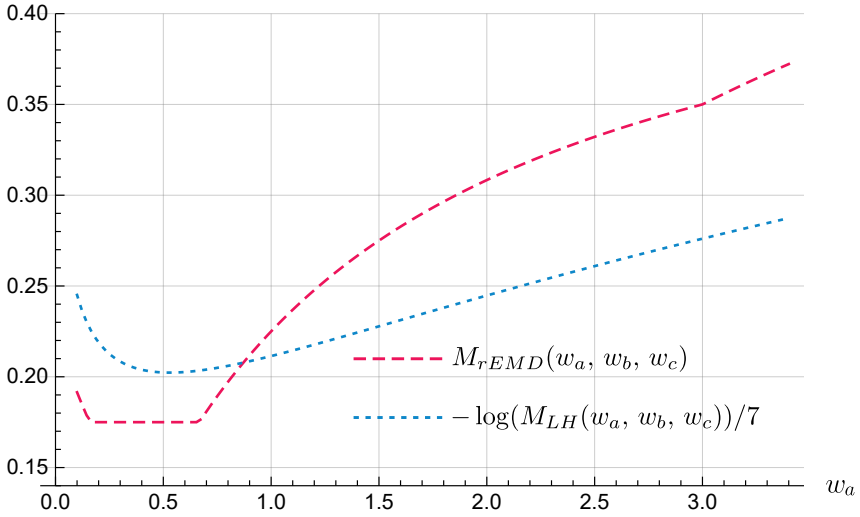


Figure 4.4 – M_{rEMD} and $log(M_{LH})$ function between L_3 and S_3

Accordingly, the rEMD in this case results from transferring probability mass from the trace $\langle b, c\rangle$ to $\langle a, c\rangle$, $\langle c, a\rangle$ and $\langle c, b\rangle$ in $\mathcal{S}_{S_3}$ in order to match the target distribution $\mathcal{S}_{L_3}$. The normalised Levenshtein distance between $\langle b, c\rangle$ and $\langle a, c\rangle$ is $1/2$, whereas it equals 1 between $\langle b, c\rangle$ and each of $\langle c, a\rangle$ and $\langle c, b\rangle$. These distances determine the cost of moving probability between traces, and consequently, the shape of the EMD function with respect to the model parameters. Hence, if the four inequalities are satisfied, the rEMD is given by:

$$\begin{aligned} M_{rEMD}(\overline{w}) = &\frac{1}{2}(\mathcal{S}_{L_3}(\langle a, c\rangle) - \mathcal{S}_{S_3}(\langle a, c\rangle)) \\ &+ (\mathcal{S}_{L_3}(\langle c, a\rangle) - \mathcal{S}_{S_3}(\langle c, a\rangle)) \\ &+ (\mathcal{S}_{L_3}(\langle c, b\rangle) - \mathcal{S}_{S_3}(\langle c, b\rangle)) \\ &= \frac{7}{40} = 0.175 \end{aligned}$$

which is a constant and also turns out to be the minimum (see Figure 4.4).

Second, the rEMD is not convex over the interval $w_a \in [2/3, 3]$. Third, its partial derivative with respect to w_a exhibits jump discontinuities, notably around $w_a \approx 0.16$, $w_a = 2/3$, and $w_a = 3$.

In larger models with more transitions, the three phenomena illustrated above (namely, constant regions, non-convex intervals, and jump discontinuities in the partial derivatives) become pervasive throughout the parameter space and occur frequently. These irregularities significantly affect the shape of the rEMD-based objective function and complicate the optimisation process. Indeed, our extensive numerical experiments indicate that only derivative-free optimisation methods are practically feasible for minimising the rEMD, as gradient-based techniques struggle with the non-smooth and non-convex nature of the objective landscape.

On the other hand, the log-likelihood function, as suggested by Equations (4.5) and (4.6), is a smooth function of the weights and is therefore more amenable to optimisation techniques, particularly gradient-based methods. In Figure 4.4, we also

plot the log-likelihood function, scaled by a factor of 7 to frame the problem as a minimisation task and to place its values within a range comparable to that of the rEMD. On the figure, we can note that the value of w_a that optimises $log(M_{LH})$ (about $w_a = 0.5$) optimises also M_{rEMD}. This however is not guaranteed in general.

4.3 Weight optimisation

In our framework, given a stochastic language $\mathcal{S}_L$ induced from a log L, and a corresponding mined WN N, the optimisation of the model's weights can be performed using either the log-likelihood or the rEMD as the objective measure. The resulting objective function f_{obj} is denoted by

$$f_{obj}(\mathcal{S}_L, N, \overline{w}, \texttt{measure}, \texttt{memoised}, \texttt{derivatives})$$

where

- $\overline{w}$ is the initial vector of weights to be optimised that need to be associated with N to make it an sWN S,
- `measure` specifies whether the log-likelihood or rEMD is used to measure the distance between the stochastic language returned by the unfolding procedure applied to the sWN S and the stochastic language of the log $\mathcal{S}_L$, that is, if `measure` is equal to
  - "rEMD" then f_{obj} computes $M_{rEMD}(\overline{w})$ given in (4.4);
  - "LH" then f_{obj} computes $-\log(M_{LH}(\overline{w}))$ based on (4.2) where the minus sign allows us to apply minimisation in both cases;
- `memoised` is a boolean flag indicating whether to use the approach described in Section 3.2, where a new unfolding is performed for each weight vector considered during the optimisation process, or the alternative described in Section 3.3, where a single unfolding is launched in which reusable functions are generated and evaluated with memoisation.
- `derivatives` is a boolean flag, applicable only when `measure` is set to "LH". It indicates whether f_{obj} should compute the exact partial derivatives of the objective function $-\log(M_{LH}(\overline{w}))$, or whether these derivatives should instead be approximated using finite differences during the optimisation process. Note that also the sign of the derivatives discussed in Section 4.2 must be changed.

Lines 1 - 3. If the `memoised` option is selected, the unfolding setup is performed prior to any computation of the stochastic language of the parametric sWN. This setup generates the reusable functions required by the memoised unfolding during the optimisation procedure.

Lines 4 - 11. Regardless of whether the LH or the rEMD is used as the objective measure, the function f_{obj} may exhibit local optima in which the optimisation process can become trapped. To mitigate this risk, the optimisation is preceded by an

Algorithm 3 Weight optimisation

Require: `measure` $\in \{$`LH`, `rEMD`$\}$,
`memoised`, `derivatives` : boolean flags,
n_0 : positive integer (number of random initialisations),
`solver` : optimisation method (see Section 4.3.1)

Ensure: Optimised weight vector $\overline{w}_{opt}$, to be associated with N in order to obtain the optimised stochastic workflow net.

1: **if** `memoised` **then**
2: UnfoldSetup($\mathcal{S}_L$, N, $R_g(N)$)
3: **end if**

4: **for** $i = 1, 2, ..., n_0$ **do**
5: $\overline{w_i} \leftarrow$ random weights
6: $M \leftarrow f_{obj}(\mathcal{S}_L, N, \overline{w}_i,$ `measure`, `memoised`, `derivatives`$)$
7: **if** $i = 1$ or $M < M_{best}$ **then**
8: $M_{best} \leftarrow M$
9: $\overline{w}_{init} \leftarrow \overline{w_i}$
10: **end if**
11: **end for**

12: $f \leftarrow f_{obj}(\mathcal{S}_L, N, \overline{w}_{init},$ `measure`, `memoised`, `derivatives`$)$
13: $\overline{w}_{opt} \leftarrow$ Minimise(f, `solver`)
14: **return** $\overline{w}_{opt}$

initialisation phase, during which n_0 random points are sampled from the parameter space. Among these, the most promising starting point (i.e., the one yielding the lowest objective value) is selected to initialise the optimisation procedure.

Lines 12 - 14. There is an additional parameter beyond those discussed so far, namely `solver`, which specifies the optimisation method to be used. The available options for solvers are presented in Section 4.3.1. The optimisation procedure itself is performed by invoking the function MINIMISE, which takes as input the objective function f_{obj}, the initial weight vector $\overline{w}_0$, the additional parameters `measure`, `memoised` and `derivatives`, and the selected optimisation method `solver`.

In the practical implementation, the number of iterations performed by the solver can be controlled via an additional parameter, max_{iter}, in Algorithm 3, allowing the user to balance runtime against solution accuracy explicitly. In the general formulation of our framework, however, we leave the stopping criterion to the discretion of the solver, which typically relies on internal mechanisms based on convergence checks or improvement thresholds.

4.3.1 Optimisation methods

The implementation we developed (see Algorithm 3) relies on the MINIMISE function provided by the `SciPy` library in Python. This function supports a wide range of solvers, including both derivative-free and gradient-based methods, and also allows integration with user-defined solvers. However, not all solvers are suitable for our setting, due to the specific properties of the objective function and the availability of derivatives. In the following, we review the applicable solvers, highlighting their respective strengths and weaknesses in order to guide the reader toward an informed and appropriate choice:

Boundary constraints. Solvers used for optimising transition weights must be capable of handling boundary constraints, ensuring that all weights remain strictly positive throughout the optimisation process. Indeed, negative weights are not admissible in the semantics of stochastic workflow nets and would have no meaningful interpretation. A transition with a negative weight would lack any probabilistic meaning, as it would imply a negative likelihood of occurrence. It could further lead to numerical instabilities such as division by zero when normalising transition probabilities. This issue is even more evident when a weight is equal to zero, as it would imply that a transition has no likelihood of ever being fired. Such a situation is meaningless: if a transition is included in the model, it must appear in at least one trace of the log, and therefore cannot be assigned a zero probability. To further improve efficiency and obtain more interpretable weight vectors, we also impose an upper bound on the weights. In our experiments, restricting the weights to the interval $[0.001, 1]$ proved to be a practical and effective choice. This range preserves substantial expressiveness, for instance, one weight can still be up to 1000 times larger than another, while avoiding extreme values that may impair convergence or interpretability. These bounds can be easily adjusted in the implementation by the user to accommodate the characteristics of a specific model or application domain.

Gradient-based or derivative-free. Another important factor in selecting a solver is the nature of the objective function, particularly when the optimisation criterion is the rEMD. As discussed in Section 4.2, the rEMD-based objective function is non-differentiable and exhibits intricate dependencies on the model parameters. Consequently, only derivative-free optimisation methods are applicable, as they do not rely on gradient information. Such methods are particularly suitable for objective functions that are non-smooth, noisy, or lack an analytical gradient, properties that accurately characterise the rEMD in our context. In contrast, when optimising log-likelihood, the use of gradient-based methods can be advantageous, as such methods generally perform better on differentiable objective functions. Moreover, the efficient computation of first-order derivatives of the log-likelihood has been implemented within the log-driven unfolding procedure, enabling solvers to exploit gradient information during optimisation directly.

Among the solvers supported by the MINIMISE function of the `SciPy` package, four are able to handle boundary constraints. Two of them are gradient-based methods, while the other two are derivative-free:

- the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm [23] (L-BFGS-B),
- the truncated Newton conjugate method [61] (TNC),
- the Nelder-Mead simplex algorithm [33] (Nelder-Mead), and
- the Powell's conjugate direction method [64] (Powell).

On the one hand, Powell's method operates by iteratively minimising the objective function along a set of search directions. These directions are updated dynamically at each iteration based on the progress observed, without requiring gradient information. The method is deterministic and can converge rapidly for smooth, low-dimensional problems. However, it may struggle with noisy or highly non-convex functions, where the search directions become less informative. The Nelder-Mead simplex method, in contrast, explores the search space by manipulating a polytope (a simplex) through geometric operations such as reflection, expansion, contraction, and shrinkage. Its strength lies in its robustness: it can handle discontinuities, plateaus, and non-differentiable surfaces. However, the method does not guarantee convergence to a global optimum and can be sensitive to the scaling of the objective function and the dimensionality of the parameter space. Both Powell's method and Nelder-Mead are therefore well-suited to our context when optimising non-differentiable objectives, such as the restricted Earth Mover's Distance (rEMD). Their main limitation is computational efficiency, as derivative-free optimisation generally requires more evaluations of the objective function.

On the other hand, L-BFGS-B and TNC are gradient-based optimisation methods specifically designed for smooth and differentiable objective functions. L-BFGS-B is a limited-memory quasi-Newton method that builds up an approximation of the inverse Hessian matrix using only a few vectors of past gradient information. This makes it especially effective in high-dimensional problems where storing or invert-

ing the full Hessian would be infeasible. It is well known for its fast convergence and robustness in large-scale optimisation under box constraints. The TNC algorithm also addresses large-scale constrained optimisation by approximating the Newton step through conjugate-gradient iterations, rather than explicitly forming the Hessian. This allows it to capture curvature information more accurately than quasi-Newton methods while remaining scalable. Its performance strongly depends on the conditioning of the problem: in well-scaled differentiable landscapes, it converges reliably, but in ill-conditioned scenarios, it may require careful parameter tuning. In both algorithms, the gradient (i.e., the vector of partial derivatives) can either be approximated numerically through finite differences or computed analytically when available. In our framework, this exact computation can be provided by the unfolding procedure, which allows the solvers to exploit precise derivative information and thus significantly improves optimisation efficiency.

4.4 Impact of the log and the associated mined WN on the complexity of the framework

Before presenting the prototype tool that has been implemented and tested, and in order to illustrate the complexity aspects of our approach with concrete examples, we first describe the event logs and the corresponding workflow nets (WNs) used in our experiments. We considered a diverse set of real-life event logs, most of which originate from the Business Process Intelligence Challenge (BPIC). All logs are publicly available at `https://data.4tu.nl/`. These logs exhibit varying levels of complexity, which in turn lead to mined WNs whose reachability graphs and corresponding unfolding directed acyclic graph (DAG) differ significantly in size and structure.

4.4.1 Real-life event log characteristics

This subsection provides a detailed description of the real-life event logs used throughout this thesis as examples, experimental data, and evaluation benchmarks. While these descriptions offer valuable insights into the complexity and difficulty of each log, they provide information for a deep understanding of the experimental results. A reader primarily interested in an overview may refer directly to Table 4.1. The table summarises the main characteristics of each event log, including the number of unique traces and activities, the average trace length, and the length of the longest trace, which together provide a concise picture of the relative complexity of each dataset. The average trace length reported is computed over the set of unique traces only. For instance, although the `Roadfines` log contains 150,370 cases, it includes only 231 distinct traces and averaging their lengths yields 8.2. Computing the average over all cases instead would give 3.7, but this value is less appropriate in our setting because the unfolding operates on unique traces rather than on cases. Using the unweighted average therefore avoids bias toward highly frequent behaviours and better reflects the structure relevant to the unfolding.

Event log	#traces	#activities	average trace length	length of longest trace
BPIC13_c	183	4	9.9	35
BPIC13_i	1,511	4	19.2	123
BPIC13_o	108	3	6.3	22
BPIC17_ol	16	8	4.1	5
BPIC20_dd	99	17	8.8	24
BPIC20_rfp	89	19	7.9	20
Roadfines	231	11	8.2	20

Table 4.1 – Characteristics of the considered real-life event logs

BPIC 2013 The BPIC 2013 collection provides three event logs originating from a Volvo IT service company, covering processes related to incident management and problem resolution over several years. These include the closed problems dataset [83] (BPIC13_c), the open problems dataset [85] (BPIC13_o), and the incidents dataset [84] (BPIC13_i). Each dataset consists of repetitions and alternations of four main activities "Accepted", "Queued", "Completed" and "Unmatched". A typical pattern is that traces often terminate with a repetition of "Completed", although this is not a strict rule. The activity "Unmatched" does not appear in every trace and is absent from BPIC13_o. When it does occur, it appears only once (no repetitions), typically near the end of the trace. In contrast, the other activities repeat multiple times and alternate with one another. As a result, any perfectly fitting workflow net discovered from these logs displays concurrency between the three looping activities ("Accepted", "Queued", and "Completed"), together with a conflict representing the optional execution of "Unmatched", as illustrated in Figure 4.5.

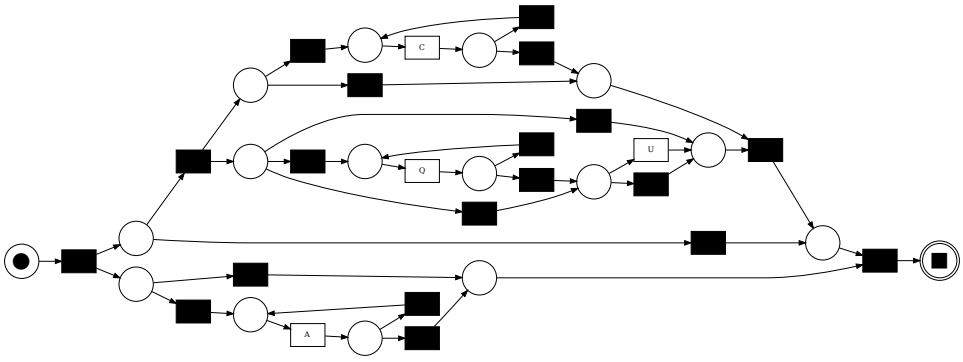


Figure 4.5 – Workflow net discovered from BPIC13_i using the Inductive Miner. Transition labels have been simplified to their initial letters.

BPIC 2017 The BPIC 2017 collection records a loan application process from a Dutch financial institution. For our experiments, we utilised the offer extension

log [80] (BPIC17_ol), which contains all applications filed through an online system in 2016 and their subsequent events up to February 1, 2017. Unlike other BPIC logs, this dataset contains a much larger number of cases but only 16 unique traces. None of these traces exhibits repetitions of activities, which makes the log and its corresponding workflow net relatively easy to manipulate. The model contains no concurrency; it only involves sequences and conflicts. Each trace begins with the two activities "Create Offer" and "Created". Afterwards, there is a conflict between sending information both by "mail and online", or "online only". Finally, the process concludes with a decision on the loan application: the case may be "refused", "cancelled", or "accepted". The corresponding workflow net, built with the Inductive Miner, is depicted in Figure 4.6. An interesting observation is that the presence of silent transitions in every conflict allows the model to produce traces that omit what appear to be mandatory pieces of information, such as whether the information was sent or the outcome of the loan application. This behaviour is likely due to the presence of noisy traces in the log.

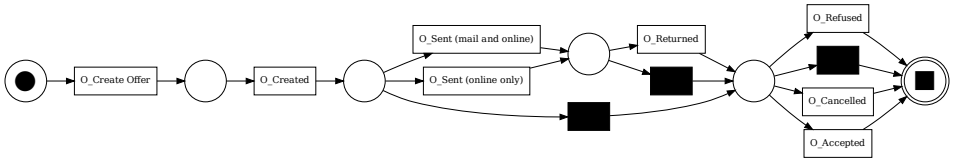


Figure 4.6 – Workflow net discovered from BPIC17_ol using the Inductive Miner

BPIC 2020 This dataset contains events about two years of travel expense claims within a Dutch university. In 2017, events were collected for two departments, while in 2018, data were recorded for the entire institution. The process involves various permits and declaration documents, all of which follow a similar flow. In this thesis, we use the sublogs corresponding to domestic declarations [78] (BPIC20_dd) and requests for payment [79] (BPIC20_rfp). Each trace follows a lengthy administrative process, resulting in huge workflow nets. Compared to the BPIC13 logs, BPIC20 contains fewer traces and is shorter in length. While BPIC13 logs express almost every possible pattern over only three or four activities, BPIC20 logs contain more structured patterns spanning 17 and 19 activities. The larger activity set and the diversity of patterns result in more complex workflow nets, as depicted in Figure 4.7, in terms of the number of transitions, places, and arcs. Consequently, the models are less general than those of BPIC13, but the number of stochastic parameters grows dramatically.

Road Traffic Fines Finally, the road traffic fine log [30] (Roadfines) is a well-known real-life event log from an information system managing traffic fines. Similar to the BPIC20 logs, Roadfines involves a large set of activities, but in this case with an even larger number of unique traces. This results in workflow nets with a correspondingly large number of nodes, as depicted in Figure 4.8 The primary source

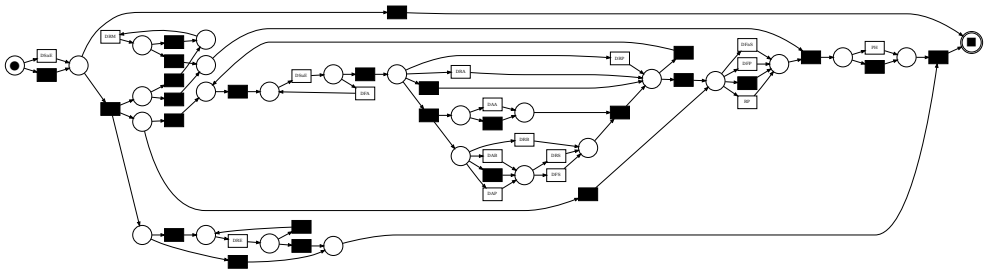


Figure 4.7 – Workflow net discovered from `BPIC20_dd` using the Inductive Miner. Transition labels have been simplified to their initial letters.

of complexity in this model lies in the size of its state space: the net contains 906 reachable markings, compared to only 235 and 247 for the BPIC20 logs.

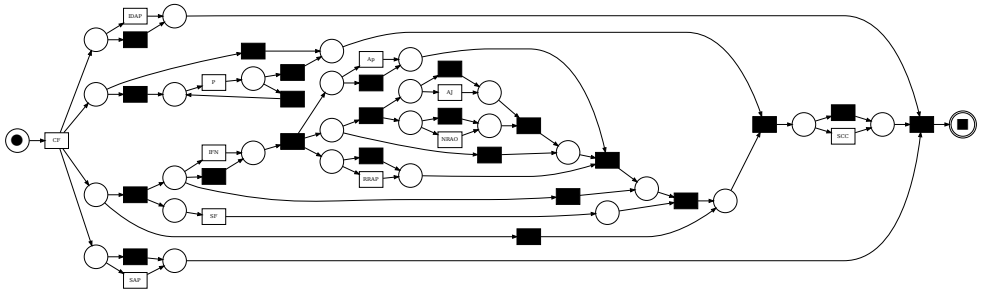


Figure 4.8 – Workflow net discovered from `Roadfines` using the Inductive Miner. Transition labels have been simplified to their initial letters.

4.4.2 From logs to net and unfolding complexity

To provide a clearer understanding of the data structures manipulated in the framework, namely the discovered workflow nets, their reachability graphs, and the unfolding DAGs, their main characteristics are summarised in Table 4.2. The table details:

- the number of transitions, which corresponds to the number of parameters that the framework must evaluate;
- the size of the reachability graph, which influences the number of nodes in the unfolding DAG and thus the elements queued and explored during the unfolding procedure;
- and the number of transition sequences, which denotes the number of distinct paths in the unfolding DAG that lead from the initial node (at level 0) to any node belonging to a $[sink]$ marking that produce a trace present in the log. This help evaluating the number of path that need to be considered in order to compute the exact, restricted to the log, stochastic language of the net. These paths are not enumerated explicitly. Different transition sequences share many common prefixes and thus traverse the same nodes and arcs

in the DAG. Our algorithm exploits this structure: the probability of reaching each node is stored and propagated once (via the queue), rather than being recomputed separately for every full path. As a result, the effective computational effort depends on the size of the unfolding (number of nodes and arcs), not on the combinatorial number of transition sequences.

Event log	#transitions	#reachable markings	#unfolding nodes	#transition sequences
BPIC13_c	19	46	16,767	$1.5{\cdot}10^{15}$
BPIC13_i	23	90	523,417	$5.8{\cdot}10^{36}$
BPIC13_o	20	74	7,504	$3.2{\cdot}10^{10}$
BPIC17_ol	11	6	27	16
BPIC20_dd	43	235	17,254	$2.6{\cdot}10^{11}$
BPIC20_rfp	51	247	45,605	$1.7{\cdot}10^{10}$
Roadfines	34	906	35,070	$4.1{\cdot}10^{11}$

Table 4.2 – Characteristics of the WNs mined from the event logs described in Table 4.1

The last two columns of Table 4.2 refer to the size of the unfolding when restricted to the traces observed in the corresponding log. For instance, in the most complex case, BPIC13_i, covering all 1,511 distinct traces from the log results in an unfolding DAG with 523,417 nodes. These traces correspond to approximately $5.8 \cdot 10^{36}$ distinct transition sequences encoded within the DAG. It is important to note that this number is not a byproduct of our unfolding approach, but rather a direct consequence of the structural complexity and behavioural richness of the mined WN itself.

Regarding the event log, the complexity of the overall problem depends in a non-trivial manner on several factors, including the number of distinct traces, the number of unique activities, and the length of the traces. The discovery algorithm used in our framework, namely, the Inductive Miner, generates a net in which each activity is mapped to exactly one labeled transition, while all other transitions are silent.

As a result, for example, the net mined from BPIC13_c contains 4 labeled transitions (corresponding to the 4 activities in the log) and 15 silent transitions, for a total of 19 transitions (see Tables 4.1 and 4.2). Silent transitions are essential to encode constructs such as concurrency, choice, and loops, which allow the WN to reproduce traces involving varied activity orderings and repeated activities. An increase in the number of activities naturally leads to more labeled transitions (one per activity), but it also tends to increase the number of silent transitions, since combining a larger set of labeled transitions into a structured and sound net requires additional control-flow logic. Furthermore, a larger number of distinct traces generally implies greater variability in the behaviour to be captured, which again tends to increase

the number of silent transitions needed to represent such diversity.

Regarding the analysis of the mined WNs, the number of reachable markings, i.e., the number of states in the reachability graph, typically increases with the number of transitions in the net. However, it is important to note that two event logs with similar statistical properties and similar numbers of transitions in their mined nets can nevertheless result in significantly different numbers of markings, due to structural differences in control-flow. The number of nodes in the unfolding DAG also depends on the lengths of the traces in the log, since longer traces require deeper exploration, thereby increasing the number of levels in the DAG. The size of the DAG plays a critical role in both the non-memoised and memoised variants of the optimisation framework in different ways. In the non-memoised approach, the DAG must be reconstructed for each new weight vector to be evaluated, which results in a substantial computational cost and contributes significantly to the overall runtime of the optimisation. In contrast, the memoised version performs the unfolding only once. However, each node in the resulting DAG must store its associated probability function, along with one additional function per transition to compute the corresponding partial derivatives. Consequently, each node stores $1+n$ functions, where n is the number of transitions. This implies that the memory requirements grow proportionally with both the number of nodes in the DAG and the number of transitions in the net.

Going into more detail on the unfolding process (Algorithm 2), we recall that the queue Q contains elements of the form $(state, level, trace)$, where elements in the queue correspond either to the same level or to two consecutive levels of the unfolding. The number of potential elements associated with a given level depends on two factors: the number of distinct markings reachable in exactly that number of transition firings (which is closely related to the total number of markings in the sWN), and the number of distinct trace prefixes that can be generated for a given marking. A larger activity alphabet typically leads to a slower unfolding process, as it increases the number of distinct trace variants to be tracked. Furthermore, the structure of the mined WN plays a critical role: the greater the number of transitions that are concurrently enabled in a given marking, the more branching occurs during the unfolding, and the slower the process becomes. In many cases, the WN allows for multiple transitions to be fired in any order from a given marking while reaching the same target marking; such symmetrical behaviour can dramatically increase the number of paths explored and thereby degrade performance. On the other hand, restricting the unfolding to traces observed in the event log typically makes the computation tractable, as it reduces the search space to only the behaviour that is actually relevant. This significantly limits the number of branches explored and helps maintain the feasibility of the approach even for complex models.

Regarding the applied distance measures, once the probabilities of the traces are known, the time required to compute $M_{LH}(\overline{w})$ as defined in Equation (4.2), or $M_{rEMD}(\overline{w})$ as defined in Equation (4.4), depends primarily on the number of distinct traces in the event log, denoted here by n. Computing the log-likelihood term $M_{LH}(\overline{w})$ has linear complexity, i.e., $O(n)$, as it involves a direct comparison between

Event log	Non-memoised unfolding	Memoised unfolding setup	Memoised computation	Memoised comp. with derivatives	LH	rEMD
BPIC13_c	$1.12 \cdot 10^{-1}$	1.22	$3.48 \cdot 10^{-2}$	$8.49 \cdot 10^{-1}$	$4.66 \cdot 10^{-4}$	$5.69 \cdot 10^{-2}$
BPIC13_i	5.93	$2.99 \cdot 10$	$6.1 \cdot 10^{-1}$	$1.95 \cdot 10$	$3.2 \cdot 10^{-3}$	5.04
BPIC13_o	$4.41 \cdot 10^{-2}$	$6.46 \cdot 10^{-1}$	$9.9 \cdot 10^{-3}$	$2.48 \cdot 10^{-1}$	$1.23 \cdot 10^{-4}$	$1.98 \cdot 10^{-2}$
BPIC20_dd	$1.1 \cdot 10^{-1}$	1.95	$1.94 \cdot 10^{-2}$	1.04	$1.53 \cdot 10^{-4}$	$1.64 \cdot 10^{-2}$
BPIC20_rfp	$3.1 \cdot 10^{-1}$	5.24	$3.42 \cdot 10^{-2}$	2.16	$2.05 \cdot 10^{-4}$	$1.42 \cdot 10^{-2}$
Roadfines	$1.74 \cdot 10^{-1}$	1.82	$5.57 \cdot 10^{-2}$	2.7	$3.52 \cdot 10^{-4}$	$1.21 \cdot 10^{-1}$

Table 4.3 – Unfolding and distance assessment computation times measured w.r.t. real-life event logs (times are given in seconds)

the probabilities of each trace in the log and those generated by the model. In contrast, computing the restrited EMD $M_{rEMD}(\overline{w})$ requires solving an optimal transport problem between two discrete probability distributions of same size n, which has a complexity of $O(n^2 \log n)$. As a result, the use of the rEMD is considerably more computationally expensive and becomes a limiting factor when dealing with logs containing a large number of distinct traces.

As for the overall optimisation procedure, the total number of transitions in the mined WN (including both labeled and silent transitions) is a critical factor, as it directly determines the number of weights to be optimised and, consequently, the dimensionality of the optimisation problem. The optimisation methods considered in our study (see Section 4.3.1) have a runtime that is proportional to two main components: the cost of evaluating the objective function (which involves computing either $M_{LH}(\overline{w})$ or $M_{rEMD}(\overline{w})$, and the number of iterations performed during the optimisation process.

4.4.3 Execution time of calculation of the stochastic language of an sWN and the distance measures

During the optimisation of the weights, the stochastic language of the mined sWN and its distance from the event log must be computed repeatedly, often thousands of times. In this section, we present the execution times associated with a single evaluation of the objective function for the considered logs. These times reflect the cost of computing the stochastic language via unfolding and applying the selected distance measure (either the log-likelihood or the rEMD). The total execution times for the complete optimisation procedure are discussed separately in Section 4.5.

The execution times for the different event logs are reported in Table 4.3, with the exception of BPIC17_o1, which is omitted due to its small size rendering the corresponding timings negligible. The second column, labeled non-memoised unfolding, indicates the time required for a single unfolding execution that computes the numerical probabilities of the traces produced by an sWN given a specific vector of transition weights. The third column, titled memoised unfolding setup, corresponds

to the execution time of a single unfolding that, instead of computing concrete probabilities, builds symbolic functions. These functions can later be used to evaluate probabilities and their derivatives for different weight vectors without repeating the unfolding. The fourth column, memoised computation, shows the time required to compute the probabilities of all traces using the functions generated during the setup phase, in combination with memoisation. Finally, the fifth column, memoised computation with derivatives, extends this by also computing the derivatives of the trace probabilities with respect to each transition weight, which is essential when applying gradient-based optimisation methods.

Performing the memoised unfolding setup phase (i.e., generating symbolic functions for later reuse) requires significantly more time than a single non-memoised unfolding that computes probabilities directly. In fact, the setup phase is approximately one order of magnitude more time-consuming. However, this initial cost is incurred only once and can yield substantial efficiency gains during optimisation. This is because evaluating probabilities using the precomputed functions (as shown in the fourth column of Table 4.3) is considerably faster than recomputing them via repeated unfoldings (second column).

The magnitude of the speed-up varies across logs. The highest improvement is observed for `BPIC13_i`, with a factor of 9.6, while the lowest is for `Roadfines`, with a factor of 3.1. The ultimate benefit of this speed-up depends on the total number of iterations performed by the optimisation algorithm, a point further discussed in Section 4.5. As expected, when using memoisation, the computation becomes slower when derivatives are also evaluated (fifth column vs. fourth column). This overhead is proportional to the number of weights in the model, which equals the number of transitions, as reported in Table 4.2.

The last two columns of Table 4.3 report the execution times required to compute the distance measure (either the log-likelihood function (4.2) or the restricted Earth Mover's Distance (4.4)) under the assumption that the probabilities of all traces are already known. These results highlight the expected performance gap between the two measures: computing the Kullback-Leibler divergence (column LH) is significantly faster than evaluating the restricted EMD (column rEMD). The higher computational cost of the rEMD becomes particularly evident on datasets with a large number of distinct traces, such as `BPIC13_i`.

4.5 Prototype tool, experiments and results

4.5.1 Prototype tool implementation

To conduct our experiments, a prototype tool leveraging the scipy.optimize package from Python as been develloped. This tool implements the optimisation procedure described in Algorithm 3, which internally relies on the DAG unfolding method presented in Algorithm 2 to search for optimal weights of a given WN model. The source code, together with the event logs and all experimental results, is publicly

available in the ProDiSt tool repository at https://github.com/DocPierro/ProDiSt.git. The main steps performed by the tool, resulting in an sWN whose stochastic language closely approximates that of the input event log, are as follows:

- importing an event log in the eXtensible Event Stream (XES) format;
- discovering a WN from the log using the Inductive Miner algorithm as described in [49];
- optimising the transition weights of the mined WN by minimizing either the log-likelihood or the rEMD.

For users who do not require the complete optimisation workflow, the tool also provides standalone scripts that, given an sWN or its reachability graph, together with a weight vector, perform a single unfolding to construct the unfolding DAG (as illustrated in Figure 3.3 and 3.6) and compute the corresponding stochastic language. In addition, the scripts include functionality to compute either the log-likelihhod or the rEMD between the stochastic language of the sWN and that of a given event log.

If the objective is to optimise the weights, the tool offers several configurable parameters that allow users to tailor their experiments. Specifically, users can choose:

- the optimisation method: among L-BFGS-B, TNC, Powell, or Nelder-Mead;
- the distance measure to minimize;
- whether to use non-memoised unfolding (i.e., a new unfolding for each evaluation) or memoised computation (i.e., function generation with reuse);
- whether to use exact or approximate derivatives (only applicable when minimizing LH with gradient-based solvers such as L-BFGS-B and TNC).

4.5.2 LH-driven optimisation experiments

Table 4.4 presents the results of the experiments aimed at minimizing the log-likelihood distance. Since the objective function, $-\log(M_{LH}(\overline{w}))$, is differentiable (as shown in Section 4.2.1), both derivative-free solvers (such as Nelder-Mead and Powell) and gradient-based solvers (such as L-BFGS-B and TNC) can be employed, unlike in the case of rEMD minimization. Concerning the unfolding strategy, all three computation modes can be used:

1. non-memoised,
2. memoised without exact derivatives, and
3. memoised with exact derivative computation.

However, the third mode, memoised computation of exact derivatives, is only applicable in conjunction with solvers that exploit gradients (i.e., L-BFGS-B and TNC), as derivative-free solvers cannot benefit from such information.

Event log	Solver	Unfolding without exact derivatives (with and without memoisation)					Unfolding with exact derivatives and memoisation			
		LH	rEMD	time	time memo	#iter	LH	rEMD	time	#iter
BPIC13_c	L-BFGS-B	3.59578	0.08143	913.1	274.7	707	**3.59206**	0.08045	584.3	1160
	TNC	3.75525	0.06595	236.5	71.6	24	3.60697	0.07969	95.9	23
	Powell	3.65532	0.07553	196.9	58.5	11	N/C	N/C	N/C	N/C
	Neilder-Mead	3.71574	0.09593	239.1	71.0	2962	N/C	N/C	N/C	N/C
BPIC13_i	L-BFGS-B	8.56934	0.23281	26318.8	3507.4	324	**8.56932**	0.23285	4764.6	296
	TNC	9.49600	0.27160	19294.3	2309.6	33	8.57968	0.23292	3101.7	40
	Powell	8.57292	0.23291	5663.4	711.3	5	N/C	N/C	N/C	N/C
	Neilder-Mead	8.59385	0.232842	15831.7	2003.0	3715	N/C	N/C	N/C	N/C
BPIC13_o	L-BFGS-B	**3.73605**	0.09615	277.2	68.0	482	3.73611	0.09613	122.9	645
	TNC	3.83865	0.09711	98.0	24.7	28	3.73668	0.09603	33.2	27
	Powell	3.77081	0.09961	165.2	42.5	21	N/C	N/C	N/C	N/C
	Neilder-Mead	3.74692	0.09354	95.1	23.5	3193	N/C	N/C	N/C	N/C
BPIC17_o1	L-BFGS-B	**1.96840**	0.06388	0.041	0.021	27	**1.96840**	0.06388	0.020	27
	TNC	1.97160	0.06044	0.102	0.053	14	**1.96840**	0.06388	0.020	9
	Powell	1.96841	0.06391	0.041	0.022	3	N/C	N/C	N/C	N/C
	Neilder-Mead	1.97331	0.06403	0.123	0.067	1096	N/C	N/C	N/C	N/C
BPIC20_dd	L-BFGS-B	3.18660	0.08430	918.6	167.6	302	3.17347	0.08488	979.2	1514
	TNC	4.53316	0.11375	1174.9	210.5	53	**3.16088**	0.08488	272.4	65
	Powell	3.26755	0.08407	432.2	78.4	10	N/C	N/C	N/C	N/C
	Neilder-Mead	4.72365	0.15465	544.5	97.4	7547	N/C	N/C	N/C	N/C
BPIC20_rfp	L-BFGS-B	4.43090	0.13515	2521.8	277.5	254	4.21441	0.11718	1353.0	1046
	TNC	7.46207	0.24958	3947.0	429.1	46	**4.21367**	0.11717	471.3	56
	Powell	4.22496	0.11716	1341.4	147.3	9	N/C	N/C	N/C	N/C
	Neilder-Mead	6.16103	0.22893	1783.2	196.4	9108	N/C	N/C	N/C	N/C
Roadfines	L-BFGS-B	3.07376	0.12904	1475.7	491.3	411	3.06259	0.13020	3367.7	2353
	TNC	3.55017	0.09558	1104.1	371.1	38	**3.06127**	0.13001	476.9	47
	Powell	3.06735	0.12349	314.7	103.7	6	N/C	N/C	N/C	N/C
	Neilder-Mead	3.50365	0.10738	673.0	215.9	5614	N/C	N/C	N/C	N/C

Table 4.4 – Outcomes of LH-driven optimisation experiments (bold values denote the smallest LH measure obtained for each log)

The starting point for each optimisation experiment was selected as the weight vector $\overline{w}$ among 100 randomly generated candidates that yielded the lowest value of the objective function $-\log(M_{LH}(\overline{w}))$. For each experiment, Table 4.4 reports in the column labelled "LH" the minimum value $-\log(M_{LH}(\overline{w}opt))$ obtained at the end of the optimisation process, and in the column labelled "rEMD" the corresponding restricted Earth Mover's Distance $M_{rEMD}(\overline{w}_{opt})$ for the optimised weight vector $\overline{w}_{opt}$. In the columns labelled "time", we also report the total execution time (in seconds) required to complete the optimisation, distinguishing between experiments performed with and without memoisation.

The variations observed across different solvers and unfolding techniques highlight that the choice of optimisation method and unfolding strategy significantly affects both the accuracy of the minimization and the overall execution time.

The first key observation is that memoised computation methods consistently lead to significantly lower execution times compared to the non-memoised approach. In most cases, the reduction is substantial, with speed-ups reaching a factor of 10 or more. Moreover, even when the number of iterations is relatively low (as shown in the "iter" column of Table 4.4), the initial overhead of setting up the memoised unfolding (see Table 4.3) is quickly amortized. For instance, in the case of `BPIC17_o1` using Powell's method, a mere three iterations are sufficient to already yield a noticeable execution time improvement with the memoised version.

Regarding memoised unfolding with derivatives, this option is meaningful only for optimisation methods that can exploit gradient information. In our framework, these are L-BFGS-B and TNC. Both solvers can either approximate the gradient numerically using the objective function (see column "time memo" without exact derivatives), or use a dedicated function, if available, to compute the exact gradient (column "time" with exact derivatives). In some cases, using exact derivatives results in longer execution times. This is often due to the fact that the optimiser performs significantly more iterations when exact gradient information is available For example, in the case of `BPIC13_c` with L-BFGS-B, optimisation with exact derivatives takes 1160 iterations (584.3 seconds), while the version using approximate derivatives requires only 707 iterations (274.7 seconds). This difference arises from the stopping criteria employed by the optimiser: minimization halts either when the improvement in the objective function becomes smaller than a predefined threshold, or when the projected gradient norm becomes too small. With exact derivatives, the optimiser more precisely identifies descent directions, which typically leads to more iterations and a more accurate local minimum.

However, in the case of L-BFGS-B, this higher precision does not always yield significantly better results. For example, for `BPIC20_rfp`, the final LH value only improves slightly from 4.43 to 4.21. In contrast, TNC benefits more consistently from exact derivatives. In several cases, including `BPIC20_rfp`, the improvement is substantial, with the final LH value decreasing from 7.46 to 4.21, highlighting the advantage of using analytical gradients in conjunction with this solver.

Concerning the derivative-free solvers, Powell and Nelder-Mead, it is notewor-

thy that neither of them ever yields the best value of the objective function. Among the two, Powell generally outperforms Nelder-Mead in terms of the obtained minimum, although its runtime performance is more variable: it can be either significantly faster or slower depending on the dataset. In terms of execution time, both solvers tend to be faster than the gradient-based methods L-BFGS-B and TNC in many scenarios, particularly for high-complexity datasets such as `Roadfines`. This makes them an appealing option when a moderate-quality solution is sufficient and computational efficiency is a priority.

To summarize the findings derived from Table 4.4:

- L-BFGS-B demonstrates consistent performance across all event logs and unfolding methods. It often achieves the best objective values but does not significantly benefit from using exact derivatives in terms of optimisation outcome or execution time.

- TNC, on the other hand, shows a strong dependence on exact derivatives to achieve optimal results and generally exhibits lower execution times compared to L-BFGS-B.

- Powell and Nelder-Mead, the two derivative-free solvers, yield slightly inferior objective values overall but often outperform gradient-based methods in terms of runtime, particularly on high-complexity datasets.

4.5.3 rEMD-driven optimisation experiments

Table 4.5 presents the results of minimizing the rEMD across the same event logs, using only derivative-free solvers, since gradient-based methods are not applicable in this context. As for the LH-driven minimisation experiments, the starting point of each optimisation was selected from among 100 randomly generated candidates. As expected, memoised computation significantly improves execution time here as well, often by an order of magnitude or more.

In several cases, the optima found by the two methods differ significantly, with Powell consistently outperforming Nelder-Mead when such differences occur. For instance, in the case of `BPIC20_rfp`, Powell achieves an optimum of 0.0486, whereas Nelder-Mead reaches only 0.416. In terms of execution time, however, Nelder-Mead is consistently faster across all cases. When model complexity becomes excessively high, as with `BPIC13_i`, optimising using rEMD becomes infeasible. This is due to the excessive computation time required for a single rEMD evaluation (about 5 seconds for `BPIC13_i`, as shown in Table 4.3), which needs to be repeated numerous times during optimisation.

Comparing Tables 4.4 and 4.5, we observe, as expected, that the LH values obtained in Table 4.4 are systematically lower than those in Table 4.5, while conversely, the rEMD values in Table 4.5 are consistently better than those reported in Table 4.4.

Event log	Solver	Unfolding without and with memoisation				
		LH	rEMD	time	time memo	#iter
BPIC13_c	Powell	7.073	0.050	602.83	395.60	13
	Neilder-Mead	5.465	**0.047**	305.46	198.02	1715
BPIC13_i	Powell	T/O[1]	T/O	T/O	T/O	T/O
	Neilder-Mead	T/O	T/O	T/O	T/O	T/O
BPIC13_o	Powell	4.0200	**0.0721**	153.3	89.2	10
	Neilder-Mead	3.99322	0.0722	73.2	42.3	1259
BPIC17_ol	Powell	2.62802	0.01676	1.007	0.918	10
	Neilder-Mead	2.78775	**0.01669**	0.479	0.433	816
BPIC20_dd	Powell	9.19973	**0.01140**	2047.0	656.1	33
	Neilder-Mead	13.61833	0.22279	343.5	105.5	3600
BPIC20_rfp	Powell	7.92734	**0.04860**	8925.1	1389.5	50
	Neilder-Mead	20.00176	0.41604	1982.1	296.92	9183
Roadfines	Powell	7.37742	**0.02560**	1983.1	1256.9	18
	Neilder-Mead	5.50840	0.04878	956.1	605.8	4076

Table 4.5 – Outcomes of rEMD-driven optimisation experiments using derivative-free methods

4.5.4 Consistency of the optimisation outcome w.r.t. the starting point

The optimisation engines require an initial vector of transition weights to begin the search for optimal parameter values. As described in Algorithm 3, the optimisation starts from the best candidate selected among n_0 randomly generated initial vectors. To assess the impact of the starting point on the final outcome, we condct experiments where we evaluate the results obtained when each of the n_0 random vectors is used individually as the initialization. We report these results for two small representative event logs: `BPIC17_o1` and `BPIC13_o`. For each log, we perform a series of LH-driven optimisations using L-BFGS-B, and another series of rEMD-driven optimisations using Powell. Each figure illustrates the outcomes of the n_0 optimisation runs. In these plots, the X-coordinate corresponds to the value of the objective distance measure before optimisation (i.e., for the initial vector), while the Y-coordinate corresponds to the optimised value of the same distance measure. The selected set of initial vectors covers a wide range of starting values in order to capture diverse optimisation behaviours. For each figure, we compute and plot both the mean (μ) and the standard deviation (σ) of the final distances. A point is considered an outlier if its value deviates by more than 3σ from the mean μ.

Figure 4.9 gives us the result of these experiments on the log `BPIC17_o1`, both on LH-driven at the top and rEMD-driven at the bottom. On this example, both optimisation strategies demonstrate fairly consistent behaviour. Most of the points lie within one standard deviation (σ) from the mean of the final distance (μ), with only a few outliers exceeding three standard deviations. More specifically, the majority of the non-outlier point are all really close to the Y-axis mean μ. Moreover, these plots offer valuable insight into the effectiveness of the optimisation framework by clearly showing the improvement between the initial and final distances. This improvement is particularly striking in the case of rEMD-driven optimisation on the `BPIC17_o1` log, where an over 26-fold reduction in the rEMD is observed (e.g., point $(0.40819, 0.0154)$). For the same log, the LH-driven optimisation shows more than a threefold improvement in the LH value (e.g., point $(7.34712, 1.96839)$). Although the magnitude of improvement is apparent, interpreting the improvement in LH is less straightforward due to the unbounded nature of the LH measure.

Figure 4.10 give us the same result, this time for `BPIC13_o`. We observed that this time, when it comes to LH-driven optimisation, the series of points expends a lot around the Y-axis mean μ but each of them stay in a small range not enough for them to be outliers. Figures further shows that selecting as starting point the weight vector with the best initial distance consistently leads to one of the best final outcomes. This confirms the effectiveness of the initialization strategy adopted in Algorithm 3, where n_0 candidates are evaluated prior to the optimisation. Moreover, the standard deviation of the final distances (i.e., the σ values on the Y-axis) remains remarkably low across all cases, indicating that the optimisation procedure yields highly stable and reproducible results regardless of the initial point, provided it is selected among a reasonable number of candidates.

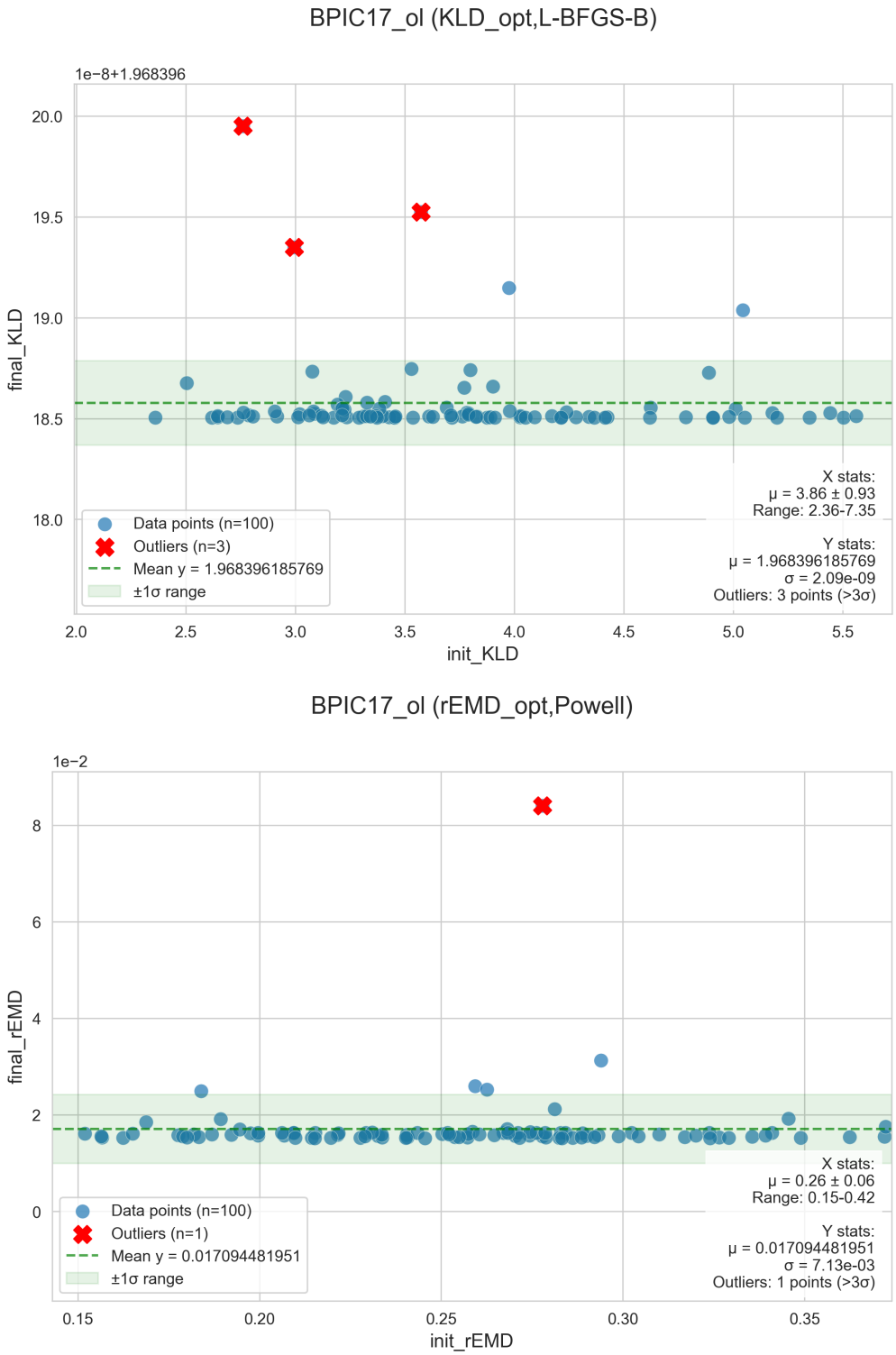


Figure 4.9 – Consistency of LH-driven (top) and rEMD-driven (bottom) optimisation for the `BPIC17_ol` log

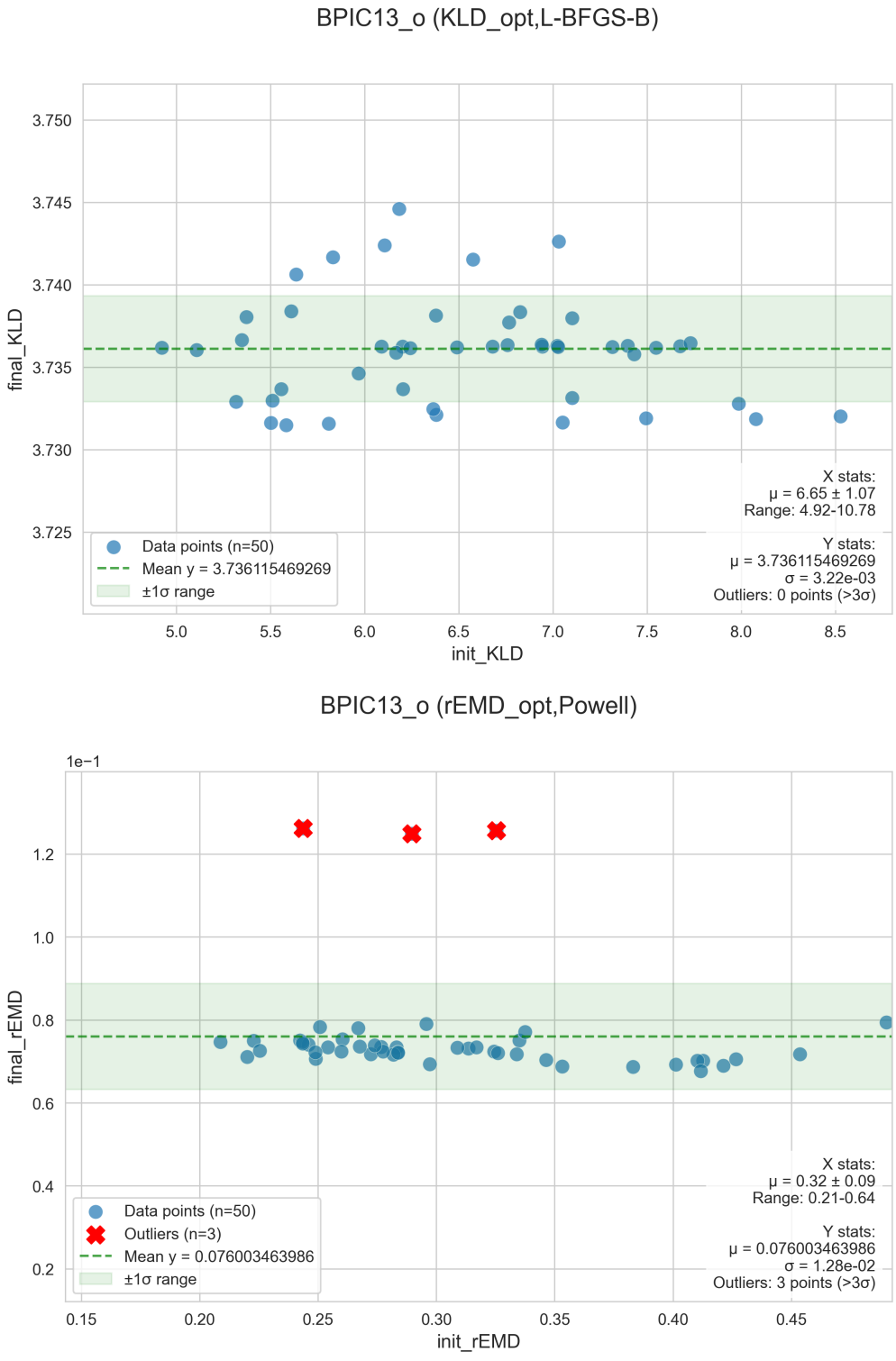


Figure 4.10 – Consistency of LH-driven (top) and rEMD-driven (bottom) optimisation for the `BPIC13_o` log

We also ran the same experiment on the `BPIC13_c` log, which led to the same conclusions as before. The results are provided in Appendix B.

4.5.5 Comparison with alternative methods

The problem of discovering stochastic process models from event logs has recently attracted increasing attention, leading to the development of a variety of approaches. In this section, we provide a comparative evaluation of our indirect, unfolding-based optimisation framework against prominent alternative indirect techniques. In particular, we consider:

- the weight estimation framework of Burke *et al.* [21],
- the Wasserstein-based Weight Estimation (WaWE) method [17], and
- the SLPN Miner proposed by Leemans *et al.* [45].

Although all these approaches aim at deriving a stochastic model that captures the empirical distribution of traces observed in an event log, they rely on fundamentally different estimation principles and computational strategies. The objective of this comparison is therefore to highlight their respective strengths and limitations, with a particular focus on accuracy, scalability, and applicability to real-life logs.

The results of these comparative experiments are reported in two separate tables. Table 4.6 compares our unfolding-based framework with the weight estimation approach and the WaWE framework. In contrast, Table 4.7 presents the comparison with the SLPN Miner. This separation is motivated by a fundamental difference in modelling assumptions. Both our framework, Burke's estimators, and WaWE operate under the assumption of perfectly fitting workflow nets. Such nets are constructed to reproduce exactly all traces observed in the log. The SLPN Miner, on the other hand, follows a different strategy: it relies on either Directly Follows Model Mining [54] (DFMM) or the Inductive Miner infrequent [50] (IMf), both of which may discover underfitting models. These models do not necessarily reproduce all traces of the event log and can deliberately abstract from, or generalise, infrequent behaviours. Consequently, the assumptions and objectives of the SLPN Miner differ substantially from those of the other methods, which justifies its separate evaluation. In this comparative study, we applied the following evaluation protocol consistently across all considered methods and event logs:

1. We first discovered a WN model using the control-flow discovery algorithm associated with each method: the standard Inductive Miner for Burke's estimators and the WaWE framework, and the Inductive Miner infrequent (IMf) for the SLPN Miner.
2. Each method was then applied to determine the transition weights of the mined WN, thereby yielding an sWN.
3. Finally, we assessed the quality of the resulting sWN by computing LH and the rEMD between its stochastic language and that derived from the event log.

Event log	Unfolding based LH optimisation (L-BFGS-B method)		Unfolding based rEMD optimisation (Powell method)		Burke's estimation [21]			WaWE [17]	
	LH	rEMD	LH	rEMD	Best estimator	LH	rEMD	LH	rEMD
BPIC13_c	**3.59578**	0.08143	7.073	**0.050**	fork	9.84876	0.63292	12.01422	0.20864
BPIC13_i	**8.56934**	**0.23281**	T/O	T/O	lhpair	15.21899	0.70457	48.52248	0.69046
BPIC13_o	**3.73611**	0.09613	4.0200	**0.0721**	pairs	5.21599	0.24232	8.22025	0.26332
BPIC17_o1	**1.96840**	0.06388	2.62802	**0.0167**	freq	5.80919	0.09871	2.12039	0.08415
BPIC20_dd	**3.18660**	0.08430	9.19973	**0.0114**	fork	25.13588	0.93164	31.15405	0.62291
BPIC20_rfp	**4.43090**	0.13515	7.92734	**0.0486**	freq	97.09419	0.98946	52.38982	0.97665
Roadfines	**3.07376**	0.12904	7.37742	**0.0256**	pairs	5.99452	0.26971	26.76204	0.47742

Table 4.6 – Comparing unfolding based optimisation with Burke's estimators and the WaWE framework

Comparison with the Burke's estimation. The group of columns denoted "Burke's estimation" in Table 4.6 reports, for each log, the best LH and rEMD scores obtained among all six estimators, and compares them with the results produced by the unfolding-based optimisation approach introduced in this chapter. Specifically, "freq" refers to the frequency-based estimator, "lhpair" to the left-handed activity pair estimator, "pairs" to the mean-scaled activity pair estimator, and "fork" to the fork distribution estimator. The results clearly show that, across all considered logs, our unfolding-based optimisation approach yields significantly lower values for both distance measures. This improvement is expected, since Burke's framework does not perform any search or fitting procedure. Its estimators are designed to be lightweight and computationally efficient, but are not design to minimize divergence from the empirical log distribution. Among the estimator, we didn't considered the alignment one for two reason. First, by examining the experimental results reported in [20], we observed that, for the logs we have in common (`BPIC13_c`, `BPIC13_i`, and `BPIC13_o`), the alignment estimator performed poorly when applied to the workflow nets discovered by the Inductive Miner. For `BPIC13_i` and `BPIC13_o`, the reported value for the alignment estimator was "–0", which we interpret as an error rather than a meaningful measurement. These findings suggested that the estimator was unreliable or, at minimum, inconsistent with the others for the logs under consideration. Second, we were unable to obtain an implementation of the alignment estimator from the original authors, and the technical description provided in [21] was not sufficiently detailed to allow us to reproduce the estimator unambiguously with our current level of understanding. Given both the questionable results reported in the literature and our inability to reconstruct the estimator with confidence, we concluded that including it in our evaluation would not add value and could risk misrepresenting its behaviour.

Comparison with the WaWE framework. Since WaWE depends on five hyperparameters that strongly influence the quality of the results, we relied on the reference implementation provided by the authors, using the default configuration: 800 paths sampled from the model and 800 traces from the log, random initialization of the weights (warm start disabled), and Phase II disabled. The minimi-

Event log	Unfolding based LH optimisation (L-BFGS-B method)		Unfolding based rEMD optimisation (Powell method)		SLPN Miner [45]	
	LH	rEMD	LH	rEMD	LH	rEMD
BPIC13_c	**0.18812**	0.00366	0.46544	**0.00183**	0.47738	0.16463
BPIC13_i	**0.40474**	0.20309	0.41811	**0.19798**	T/O	T/O
BPIC13_o	**0.54136**	0.03897	0.84106	**0.02173**	0.79513	0.10925
BPIC17_ol	**1.78159**	0.06163	2.29189	**0.01653**	1.89136	0.07766
BPIC20_dd	**0.80139**	$3.67 \cdot 10^{-7}$	3.68165	$\mathbf{1.97 \cdot 10^{-7}}$	0.82936	0.02778
BPIC20_rfp	**0.76657**	0.01077	4.13115	$\mathbf{1.36 \cdot 10^{-6}}$	0.78117	0.01245
Roadfines	**2.81308**	0.13955	3.62986	**0.02444**	6.20336	0.31502

Table 4.7 – Comparing unfolding based and SLPN miner optimisation on under-fitting models

sation metric selected is the penalized Earth Mover's Stochastic Conformance [17] (pEMSC), while all remaining parameters were kept at their default values. Given the variability in the quality of the output produced by WaWE, we executed the discovery method 30 times for each event log. For each run, we computed the resulting rEMD, and in Table 4.6 we report the median rEMD value, in line with the experimental protocol described in [17]. In most cases, the WaWE framework yields substantially worse rEMD values than our unfolding-based optimisation approach. Several factors can explain this performance gap:

- WaWE relies on a different variant of the Earth Mover's Distance (i.e., the penalized EMSC), which is not directly aligned with the rEMD used in our evaluation;
- the selection of appropriate hyperparameter values is non-trivial and has a considerable impact on the quality of the results; and
- WaWE evaluates only a limited number of paths in the model, which may hinder its ability to capture the full stochastic behaviour, particularly for complex or highly branching logs.

Comparison with the SLPN Miner. For the sake of simplicity, in our comparative study we focused on the IMf approach only and proceeded in two directions.

First, we attempted to run the SLPN Miner on WN models discovered with a noise threshold of 0 (corresponding to perfect fitness). With the exception of the BPIC17_ol log, which yields a very simple model, this led to a computational bottleneck: the SLPN framework was unable to construct the cross-product between the symbolic automata representing the log traces and the reachability graph of the net, which is required to compute their trace probabilities.

We then conducted experiments on WN models discovered using strictly positive noise thresholds. In this setting, the SLPN framework successfully completed the optimisation procedure for all but one of the WN models obtained with a threshold of 0.2, returning an optimised weight vector for each. Subsequently, we applied our unfolding-based optimisation framework to the same underfitting models previously processed by the SLPN Miner. It is worth noting that our unfolding procedure naturally supports underfitting WN models without requiring any modification to Algorithm 2 or Algorithm 3. The unfolding terminates once all log traces reproducible by the model have been covered, and both LH and rEMD are then computed exclusively over these reproducible traces.

Table 4.7 reports the LH and rEMD values obtained with the SLPN Miner and with our unfolding-based optimisation approach. In all cases, both optimisation procedures completed within one second of execution time. Across all considered logs, our unfolding-based approach consistently achieved lower distance values, indicating a closer alignment between the discovered stochastic model and the event log. Notably, for the most complex log in our dataset, `BPIC13_i`, the SLPN framework failed to complete the construction of the symbolic equations, even when using noise thresholds above 0.2.

To extend the comparative study with the SLPN Miner, we further investigate the impact of the noise threshold on the control-flow discovery of the Inductive Miner infrequent variant, in terms of the complexity of the resulting net. We compare perfectly fitting models (with a noise threshold of 0) to under-fitting ones (with a threshold of 0.2) across the event logs in our benchmark pool, as reported in Table 4.8 and Table 4.9.

With the exception of the `BPIC17_o1` log, all under-fitting models reproduce fewer than 5% of the traces in their corresponding logs. The impact of the noise threshold on model complexity is substantial: the number of transitions decreases by up to 38% (e.g., `Roadfines`), while the number of reachable markings drops by as much as 86% (e.g., `BPIC13_o`). More critically, noise filtering eliminates loops in 4 out of the 6 models, namely, `BPIC13_o`, `BPIC20_dd`, `BPIC20_rfp`, and `Roadfines`, whereas the corresponding non-filtered models contain loops. Since the absence of loops rules out all traces with repeated activities, the control-flow structure is drastically simplified. As a consequence, the number of nodes in the unfolding is reduced dramatically, by up to 99.993% in the case of `BPIC20_rfp`. Overall, a noise threshold of 0.2 has a profound impact on both the structure of the discovered models and the complexity of the subsequent optimisation process.

In summary, the experimental evidence discussed above highlights several aspects that warrant further consideration.

First, in order to make a fair comparison with the SLPN framework, it was necessary to apply a positive noise threshold to filter out infrequent behaviours. Following the experimental setup described in [45], we set this threshold to 0.2. Without such filtering, and the resulting simplification of the WN models, the SLPN framework is unable to complete the optimisation process.

Second, while the results reported in Table 4.7 indicate that unfolding-based optimisation yields models with lower LH and rEMD values, this does not necessarily imply that our approach produces more accurate models than the SLPN framework. Indeed, the optimisation outcome depends on the chosen objective function (LH or rEMD in our case, versus uEMSC or ER^{-1} for the SLPN optimiser). A more balanced comparison would therefore require adapting one of the frameworks to operate under the same optimisation criteria as the other.

Finally, the implications of noise filtering warrant closer examination. Although under-fitting models may, in some cases, improve stochastic conformance (as reported in [45]), the impact of noise-filtering techniques (such as IMf or DFMM) on the discovered models is far from trivial. Without a detailed analysis of the event log in question, it is difficult to determine whether the filtered traces correspond to genuine noise (e.g., low-frequency anomalies or logging errors) or to meaningful behaviour (e.g., repetitions of activities). Our experiments indicate that the latter are frequently excluded when a non-zero noise threshold is applied, thereby simplifying the models but potentially at the cost of representational accuracy.

4.6 Conclusion

In this chapter, we addressed the problem of deriving stochastic process models that faithfully reproduce the probabilistic behaviour observed in real-life systems, as captured by event logs. Unlike traditional process discovery methods, which focus solely on the control-flow perspective and disregard the relative frequencies of traces, our objective was to obtain stochastic models that can reflect the likelihood of observed behaviours.

To this end, we proposed a weight estimation framework that searches for an optimal parametrisation of a non-stochastic workflow net, thereby producing an sWN whose stochastic language closely matches the empirical distribution of the log. The framework has also been extended to compute first-order derivatives of smooth and differentiable distance measures, enabling integration with gradient-based optimisation algorithms. Similarly, second-order derivatives can be incorporated, as defined in Appendix A, making the algorithm applicable in Hessian-based optimisation methods. Extensive experiments on real-life event logs demonstrated the practical effectiveness of the proposed approach. The results highlighted not only the trade-offs between accuracy and efficiency associated with different optimisation strategies, but also the overall feasibility of aligning stochastic models with empirical observations at scale.

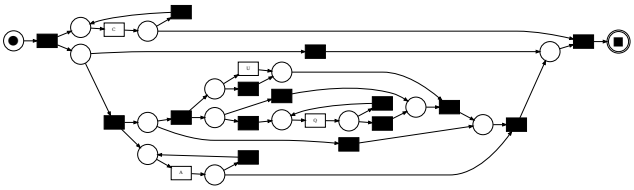
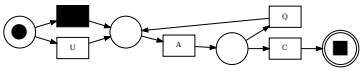
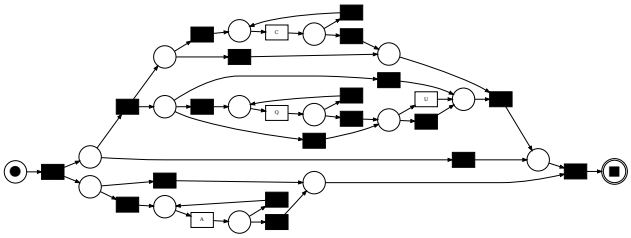
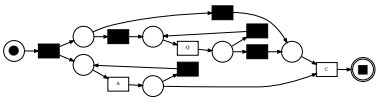
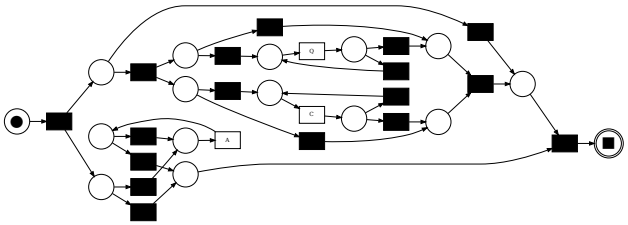
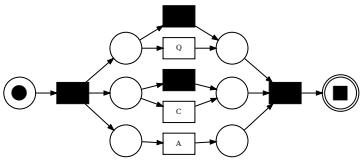
PM4PY Inductive Miner (noise=0)	ProM Inductive Miner (noise=0.2)
`BPIC13_closed` with 4 activities and 183 traces	
	
#transitions 19 | #reachable markings 46 | #unfolding nodes 16,767 | %modeled traces 100%	#transitions 5 | #reachable markings 4 | #unfolding nodes 26 | %modeled traces 2.73%
`BPIC13_incidents` with 4 activities and 1,511 traces	
	
#transitions 23 | #reachable markings 90 | #unfolding nodes 523,417 | %modeled traces 100%	#transitions 9 | #reachable markings 10 | #unfolding nodes 42,844 | %modeled traces 2.78%
`BPIC13_open` with 3 activities and 108 traces	
	
#transitions 20 | #reachable markings 74 | #unfolding nodes 7,504 | %modeled traces 100%	#transitions 7 | #reachable markings 10 | #unfolding nodes 25 | %modeled traces 4.63%

Table 4.8 – Perfectly fitting (left) and under-fitting (right) WN models discovered with the inductive miner infrequent using different noise thresholds

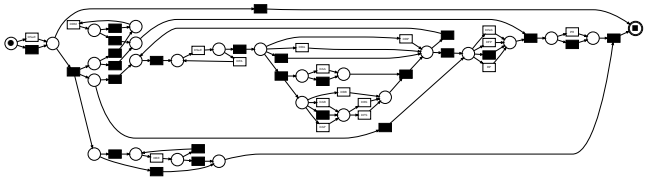
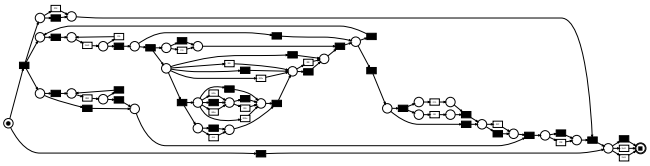
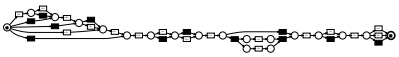
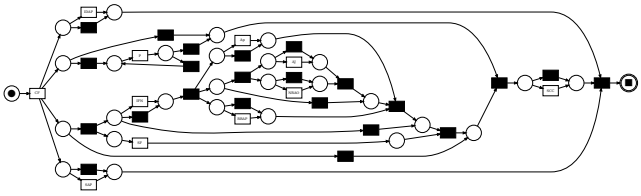
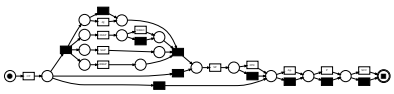
PM4PY Inductive Miner (noise=0)	ProM Inductive Miner (noise=0.2)
`BPIC20_dd` with 17 activities and 99 traces	
	
#transitions 43	#reachable markings 235
#unfolding nodes 17,254	%modeled traces 100%
#transitions 25	#reachable markings 16
#unfolding nodes 31	%modeled traces 3%
`BPIC20_rfp` with 19 activities and 89 traces	
	
#transitions 51	#reachable markings 247
#unfolding nodes 45,605	%modeled traces 100%
#transitions 29	#reachable markings 19
#unfolding nodes 32	%modeled traces 3.37%
`Roadfines` with 11 activities and 231 traces	
	
#transitions 34	#reachable markings 906
#unfolding nodes 35,070	%modeled traces 100%
#transitions 21	#reachable markings 32
#unfolding nodes 40	%modeled traces 2.6%

Table 4.9 – Perfectly fitting (left) and under-fitting (right) WN models discovered with the inductive miner infrequent using different noise thresholds

Chapter 5

Statistical Bayesian Inference for Stochastic Process Discovery

In the previous chapter, we addressed the problem of discovering stochastic workflow nets by formulating weight estimation as an optimisation task. While this approach proved effective in aligning the model with the empirical distribution of the log, it essentially delivers only the final optimised parameter vector. No information is retained on how the weights evolve during the search, and the resulting point estimate provides little insight into parameter uncertainty or the range of alternative explanations supported by the data. Moreover, exact computations required by optimisation-based techniques may become prohibitively expensive for large models, as the unfolding of the reachability graph can grow rapidly.

To address these limitations, this chapter explores a complementary simulation-based approach. We propose a framework for inferring transition weight parameters in a stochastic workflow net (sWN) model, denoted as S. The approach combines the HASL procedure, used to approximate the stochastic language $\mathcal{S}_S$ generated by S, with a tailored instance of the Approximate Bayesian Computation via Sequential Monte Carlo (ABC-SMC) inference scheme. As illustrated in Figure 5.1, the procedure begins by applying the Inductive Miner algorithm [49] to the input event log in order to discover a workflow net structure. This structure, which is guaranteed to reproduce all traces from the log, is then enriched with a vector of weight parameters $\overline{w}$. The resulting parameterised model is then supplied to the ABC-SMC engine, along with the hyper-parameters governing both inference and simulation. The outcome of the framework is a set of marginal posterior estimates for the transition weights, capturing the most plausible stochastic behaviours underlying the observed log. The remainder of this chapter is structured as follows:

- Section 5.1 introduces the HASL-based approximation of the stochastic language generated by an sWN.
- Section 5.2 presents the ABC framework for inferring transition weights.

- Section 5.3 details the estimation of the posterior distribution of the model parameters.
- Section 5.4 reports on a series of experiments on real-life event logs evaluating the effectiveness of the proposed framework.
- Section 5.5 concludes the chapter and helps position this contribution with respect to Chapter 4.

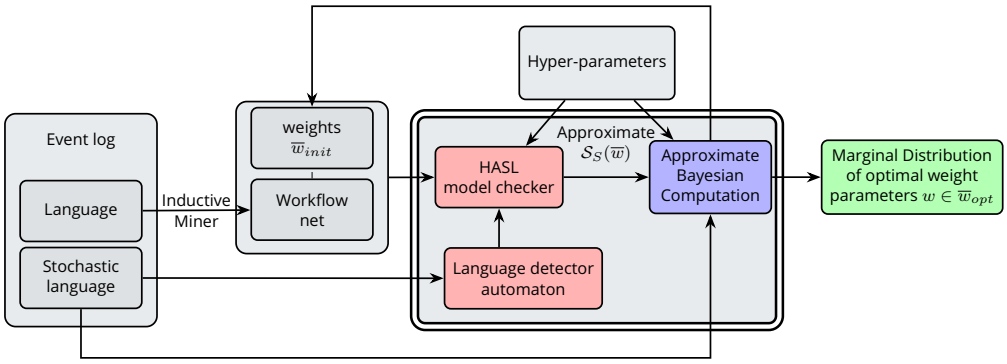


Figure 5.1 – Stochastic process discovery based on an ABC-SMC procedure

5.1 HASL-based approximation of an sWN stochastic language

The first building block of our ABC-SMC framework is the ability to efficiently estimate the stochastic language of an sWN. In contrast to the optimisation framework presented in the previous chapter, which relies on the exact computation of the language through unfolding, the ABC philosophy is rooted in simulation and statistical estimation. This shift in perspective enables us to explore a broader range of weight vectors, as the computational burden of exact evaluation is replaced by approximate yet scalable estimation. Therefore, we require an estimator that, given a weight vector, can provide a reliable approximation of the stochastic language induced by the sWN. This estimator serves as the foundation for the subsequent inference procedure.

We rely on an HASL formula (Definition 5.3) which, by combining a dedicated stochastic language detector hybrid automaton $\mathcal{A}_{sld(L)}$ (Definition 5.2) with a tailored expression Z, allows us to approximate the stochastic language $\mathcal{S}_S$ to an arbitrary degree of precision.

Since a stochastic language is composed of traces (i.e., sequences of activities over a finite alphabet) while hybrid automata operate on numerical variables, a mechanism is required to encode words as numerical values. To enable the detection of the stochastic language generated by an sWN through synchronisation with a language detector automaton, we define a mapping from words over the alphabet Σ to unique integers. This mapping must be injective, so that each distinct

word is assigned a distinct numerical value, thereby guaranteeing unambiguous identification within the automaton.

Definition 5.1 (Word mapping) Let Σ be an alphabet of cardinality n, and let

$$f_l : \Sigma \to \mathbb{N} \setminus \{0\}$$

be an injective function assigning a unique integer to each letter. We define the *word mapping* function $w_m : \Sigma^* \to \mathbb{N}$ as

$$w_m(\sigma) = \sum_{i=1}^{|\sigma|} f_l(\sigma[i]) \cdot n^{\,i-1} \tag{5.1}$$

where $\sigma[i]$ denotes the i-th activity of the word $\sigma \in \Sigma^*$. This encoding is analogous to interpreting the sequence of activity as a number written in base n.

Theorem 5.1 The word mapping function $w_m : \Sigma^* \to \mathbb{N}$, defined in (5.1), is injective: each word is mapped to a unique integer.

Proof. Let $f_l : \Sigma \to \mathbb{N} \setminus \{0\}$ be an injective mapping that assigns a distinct natural number to each activity of Σ, with $|\Sigma| = n$. For any word $\sigma \in \Sigma^*$, Definition 5.1 yields

$$w_m(\sigma) = \sum_{i=1}^{|\sigma|} f_l(\sigma[i]) \cdot n^{\,i-1}$$

Assume by contradiction that there exist two distinct words $\sigma_1 \neq \sigma_2$ such that $w_m(\sigma_1) = w_m(\sigma_2)$. Let k be the first position where $\sigma_1[k] \neq \sigma_2[k]$. Because f_l is injective, we have $f_l(\sigma_1[k]) \neq f_l(\sigma_2[k])$.

In the expression of w_m, the contribution of position k is multiplied by $n^{\,k-1}$, whereas all contributions from later positions are multiples of $n^{\,k}$. Hence, the coefficient at position k cannot be cancelled or compensated by any later term. This contradicts the assumption that $w_m(\sigma_1) = w_m(\sigma_2)$. Therefore, distinct words always map to distinct integers, and w_m is injective. □

Notice that, by Definition 5.1, the mapping of a word $\sigma \in \Sigma^*$ follows a little-endian convention: the first letter of the word $\sigma[0]$ corresponds to the least significant digit in the base-n encoding of σ. It is important to note that f_l must not assign 0 to any activity, as this would nullify its contribution to w_m and can break injectivity:

$$f_l(a) = 0 \implies w_m(\varepsilon) = w_m(\langle a \rangle) = w_m(\langle a, a \rangle) = 0$$

Example 5.1 (Word mapping).

Let $\Sigma = \{a, b, c\}$ be an alphabet, and let us assume as letters' mapping:

$$f_l(a) = 1 \qquad f_l(b) = 2 \qquad f_l(c) = 3$$

then the following are examples of mapping of words in Σ^*:

$$
\begin{aligned}
w_m(\langle a, a\rangle) &= 1 \cdot 3^0 + 1 \cdot 3^1 = 4 \\
w_m(\langle a, b\rangle) &= 1 \cdot 3^0 + 2 \cdot 3^1 = 7 \\
w_m(\langle c, b\rangle) &= 3 \cdot 3^0 + 2 \cdot 3^1 = 9 \\
w_m(\langle a, b, c\rangle) &= 1 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 = 34 \\
w_m(\langle c, b, a\rangle) &= 3 \cdot 3^0 + 2 \cdot 3^1 + 1 \cdot 3^2 = 18 \\
w_m(\langle b, a, c\rangle) &= 2 \cdot 3^0 + 1 \cdot 3^1 + 3 \cdot 3^2 = 32 \\
w_m(\langle c, a, b, b\rangle) &= 3 \cdot 3^0 + 1 \cdot 3^1 + 2 \cdot 3^2 + 2 \cdot 3^3 = 79
\end{aligned}
$$

In the remainder, given a event log L and its finite language $\mathcal{L}_L$ and a word mapping w_m (defined on alphabet Σ_L), we denote $w_m(L) \subset \mathbb{N}$ the set of naturals to which the words of the support $\mathcal{T}_L$ are mapped.

Convex remapping. Since the mapping defined in Equation 5.1 generally produces a non-convex (i.e., sparse) support set with a large supremum (exponential in the length of the word), this can negatively impact the HASL-based estimation of confidence intervals for the corresponding probability density function (PDF). In this context, we are in fact dealing with probability mass functions (PMFs), as the word mapping function we are tracking is discrete. However, we retain the term PDF to align with the model checker terminology, which uses it as a general keyword encompassing both PDFs and PMFs. To address this issue, we introduce a convex remapping scheme that maps each word of a finite log L to a value in the contiguous interval $\{0, 1, 2, \cdots, \mathcal{T}_L - 1\}$, where $\mathcal{T}_L$ denotes the set of distinct traces in L. This is achieved by first computing $w_m(L)$ offline, and then reassigning each element in $w_m(L)$ to a unique value within the target convex interval. In the remainder, for any word $\sigma \in \mathcal{T}_L$, we denote its convex remapping as

$$w_{cm}(w_m(\sigma)) \in \{0, 1, 2, \cdots, \mathcal{T}_L - 1\}$$

Definition 5.2 (Stochastic language detector automaton) Given S, an sWN discovered from an event log L with alphabet $\Sigma_L = \{a_1, \ldots, a_n\}$ ($|\Sigma_L| = n$) and given an injective mapping $f_l : \Sigma_L \to \mathbb{N} \setminus \{0\}$, the stochastic language estimator LHA $\mathcal{A}_{sld(L)} = \langle Ev, Loc, Init, Acc, X, flow, \Lambda, \to\rangle$ is defined as follows:

- $Ev = \Sigma_L \cup \{\tau\}$,
- $Loc = Init \cup Acc$, with $Init = \{l_{start}\}$ and $F = \{l_{end}\}$,
- $X = \{w, w_c, c, c_1, \ldots, c_n\}$ consisting of the following $n + 3$ (integer) variables:
  - w: mapping of detected word
  - w_c: convex mapping of detected word
  - c: length of detected word
  - c_i ($1 \leq i \leq n$): number of occurrences of letter $a_i \in \Sigma_L$ in the detected word

- Each variable has a constant rate of evolution in every location:

$$flow(l) = (0, \ldots, 0) \quad (\forall l \in Loc)$$

- $\Lambda(l_{start}) = \texttt{true}$ and $\Lambda(l_{end}) = (sink == 1)$,
- and transition set $\rightarrow$ consisting the following $n + 1$ transitions:
  - n self-loop *synchronised* transitions (with $1 \leq i \leq n$) defined as follows:

$$l_{start} \xrightarrow{\{a_i\},\, c_i \leq l_i \wedge (\sum_j c_j) \leq c_m,\, \{w \mathrel{+}= f(a_i) \cdot n^c;\, c \mathrel{+}= 1;\, c_i \mathrel{+}= 1\}} l_{start}$$

    where n, l_i and c_m are constants referred to the log L (n is the number of activity, l_i is the maximum number of the i-th activity in any trace of L, c_m is the length of the longest trace in L).
  - a self-loop *synchronised* with silent transitions defined as:

$$l_{start} \xrightarrow{\{\tau\},\, \top,\, \emptyset} l_{start}$$

  - $|\mathcal{T}_L|$ *autonomous* transitions (with $1 \leq i \leq |\mathcal{T}_L|$) defined as:

$$l_{start} \xrightarrow{\#,\, w \in w_m(L),\, \{w_c = w_{cm}(w_m(\sigma_i))\}} l_{end}$$

    where $\sigma_i \in \mathcal{T}_L$ is a unique trace of L, $w_m(\sigma_i) \in \mathbb{N}$ is its mapping and $w_{cm}(w_m(\sigma_i)) \in \mathbb{N}$ its convex remapping.

The goal of the automaton $\mathcal{A}_{sld(L)}$, depicted in Figure 5.2, is to detect, among the traces generated by the sWN S with which it synchronises, those that belong to the log L, by computing their corresponding mappings. To this end, $\mathcal{A}_{sld(L)}$ is equipped with $n + 3$ integer variables, whose goal is

- variable w stores the numeric encoding of the currently scanned trace, as defined in Definition 5.1;
- variables c_i (for $1 \leq i \leq n$) count the occurrences of each activity a_i as it appears in the trace;
- variable c keeps track of the total length of the trace, measured as the number of observed activities a_i;
- variable w_{cm} stores the convex remapping of the trace encoding, provided the trace corresponds to a word in L.

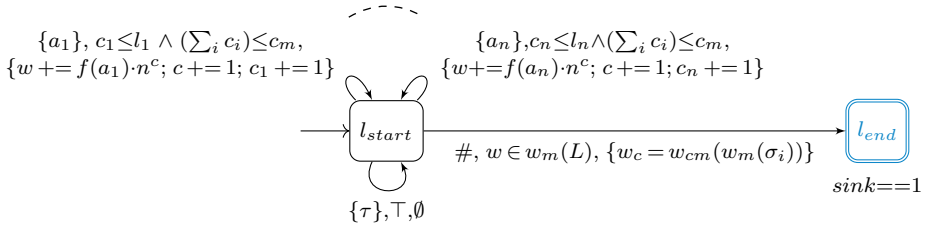


Figure 5.2 – The stochastic language detector automaton $\mathcal{A}_{sld(L)}$

Each activity in the trace is observed through the traversal of a synchronised self-loop arc on the initial location (l_{start}). Specifically, upon traversal of the self-loop associated with activity a_i, the value of w is incremented by $f_l(a_i) \cdot n^c$, and the general counter c and the corresponding activity counter c_i are increased by one.

Scanning of traces generated by an sWN model is guaranteed to terminate, either by accepting or rejecting the currently observed trace. A trace $\sigma \in \Sigma_L^*$ is accepted if and only if the following two conditions are satisfied:

1. it is produced by a sequence of transitions $\langle t_1, \ldots, t_m \rangle$ such that the final transition t_m leads the model to the deadlock marking by placing a token in the designated $sink$ place. This condition is enforced by the invariant $sink == 1$ in the final location l_{end}, and,
2. it belongs to the set of observed traces in the event log, i.e., $\sigma \in \mathcal{T}_L$. This is operationalised by the guard $w \in w_m(L)$ on transitions from l_{start} to l_{end}.

Any other trace $\sigma \in \Sigma_L^* \setminus \mathcal{T}_L$ results in the automaton blocking and, hence, it is rejected (thus its mapping value w is discarded). To ensure termination and to rule out infinite traces (particularly those arising from cycles in the net), the automaton enforces boundedness constraints during trace scanning. Specifically, each self-loop arc associated with activity a_i is only enabled if the number of observed occurrences of a_i does not exceed its maximum in the log, i.e., $c_i \leq l_i$ and the total number of observed activities does not exceed the length of the longest trace in the log, i.e., $\sum_i c_i \leq c_m$. These constraints ensure that trace detection remains finite and computationally tractable.

By construction, the automaton $\mathcal{A}_{sld(L)}$ enjoys the property that the stochastic language $\mathcal{S}_{S \times \mathcal{A}_{sld(L)}}$, generated by the product of the sWN model S and the language detector automaton $\mathcal{A}_{sld(L)}$, is contained within the set of traces $\mathcal{T}_L$ from which the model S was discovered. In other words, the automaton detects only those traces in S that are also present in the original event log. We formalise this guarantee in Theorem 5.2.

Theorem 5.2 Let S be an sWN discovered with a discovery algorithm $\mathfrak{D}$ from an event log L then

$$\mathcal{L}_{S \times \mathcal{A}_{sld(L)}} \subseteq \mathcal{T}_L$$

Proof. Straightforward consequence of HASL operational semantics. □

Lemma 5.1 If algorithm $\mathfrak{D}$ used to discover S has a perfect fitness then

$$\mathcal{L}_{S \times \mathcal{A}_{sld(L)}} = \mathcal{T}_L$$

As a consequence of Theorem 5.2, by combining the automaton $\mathcal{A}_{sld(L)}$ with the target expression $PDF(last(w), s, l, h)$, where s, l, and h are respectively the size, lower bound, and upper bound of the histogram buckets, we obtain a confidence

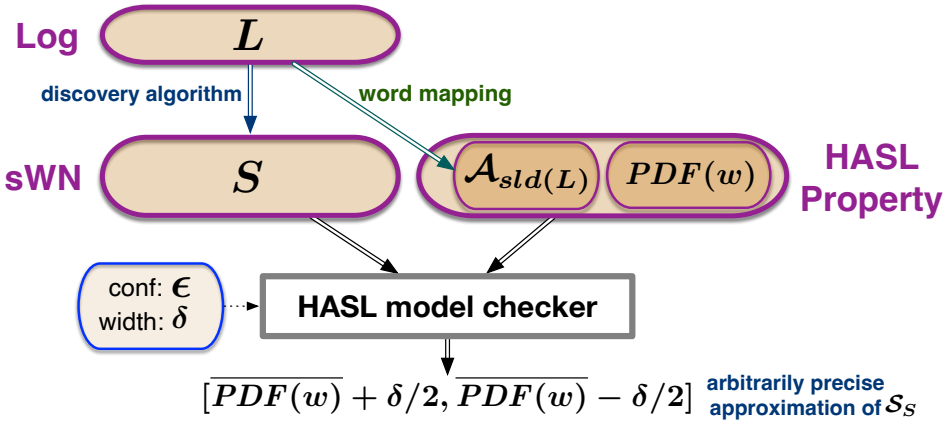


Figure 5.3 – Approximation of $\mathcal{S}_{S \times \mathcal{A}_{sld(L)}}$ via HASL model checking

interval estimator of the probability distribution with which the model S generates the traces observed in the event log L.

Definition 5.3 (HASL stochastic language estimator formula) Given S an sWN discovered from an event log L, we define the HASL stochastic language estimator formula $\varphi_{sle(L)} \equiv (\mathcal{A}_{sld(L)}, PDF(last(w_c), 1, 0, |\mathcal{T}_L| - 1))$, where $\mathcal{A}_{sld(L)}$ is the stochastic language detector automaton (Definition 5.2) and w_c its corresponding word mapping variable. Notice that since w_c is a discrete random variable in the target expression $PDF(last(w_c), 1, 0, |\mathcal{T}_L| - 1)$, we use buckets of size $s = 1$, while we use $[l, h] = [0, |\mathcal{T}_L| - 1]$ as support set for the sought approximation of the PDF of w_c.

Example 5.2 (HASL-based stochastic language estimation).

Let us consider the following stochastic language corresponding to an event log L_4 with alphabet $\Sigma = \{a, b, c\}$:

$$\mathcal{S}_{L_4} = [\langle a, b, c\rangle^{0.7}, \langle a, c, b\rangle^{0.2}, \langle a, c, b, b\rangle^{0.1}]$$

Let us assume the following letter mapping $f_l(a) = 1, f_l(b) = 2, f_l(c) = 3$ which induces the word mapping:

$$w_m(\langle a, b, c\rangle) = 1 \cdot 3^0 + 2 \cdot 3^1 + 3 \cdot 3^2 = 34$$
$$w_m(\langle a, c, b\rangle) = 1 \cdot 3^0 + 3 \cdot 3^1 + 2 \cdot 3^2 = 28$$
$$w_m(\langle a, c, b, b\rangle) = 1 \cdot 3^0 + 3 \cdot 3^1 + 2 \cdot 3^2 + 2 \cdot 3^3 = 82$$

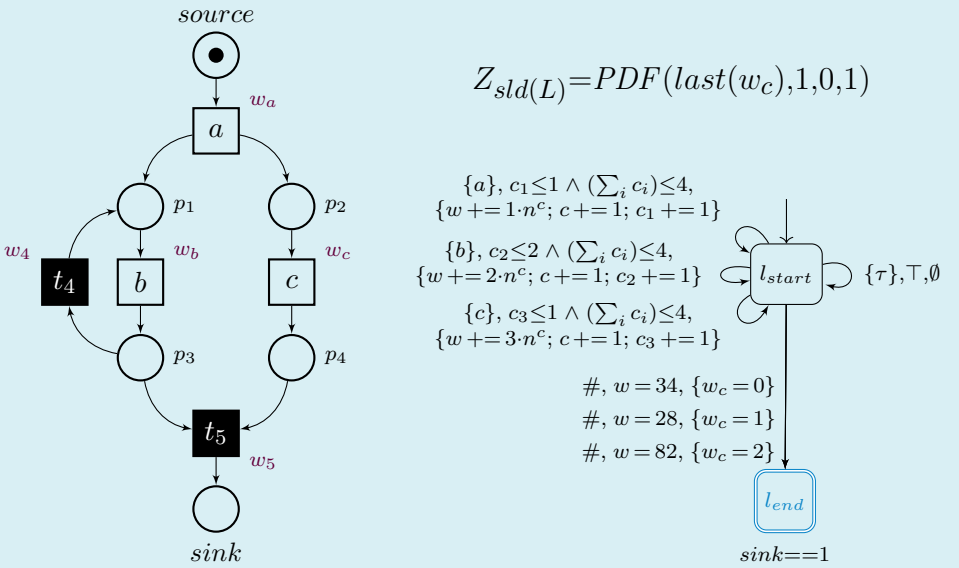
and the corresponding convex word mapping:

$$w_{cm}\big(w_m(\langle a, b, c\rangle)\big) = 0$$
$$w_{cm}\big(w_m(\langle a, c, b\rangle)\big) = 1$$
$$w_{cm}\big(w_m(\langle a, c, b, b\rangle)\big) = 2$$

Figure 5.5(a) depicts the sWN model corresponding to S_{L_4} (obtained via the Inductive Miner algorithm) while Figure 5.5(b) shows the corresponding HASL

stochastic language estimator formula $\varphi_{sle(L_4)}$ consisting of the stochastic language detector automaton $\mathcal{A}_{sld(L_4)}$, which is based on alphabet Σ and convex word mapping w_{cm} (described above) and the target formula $Z_{sld(L_4)}$.
We then apply the language-detector procedure to synchronize S with $\mathcal{A}_{sld(L_4)}$ and detect the occurrences of the three log traces in executions of S_4, thereby enabling the estimation of $\mathcal{S}_S$ restricted to:

$$\mathcal{T}_L = \{\langle a, b, c\rangle, \langle a, c, b\rangle, \langle a, c, b, b\rangle\}$$



(a) sWN S_4

(b) HASL formula $\varphi_{sle(L_4)} \equiv (\mathcal{A}_{sld(L_4)}, Z_{sld(L_4)})$

Figure 5.5 – Model and detector used in HASL

Based on this setting, one possible synchronized run is:

$$\big(\overbrace{[source]}^{m_0},\ l_{\text{start}},\ \overbrace{[0,0,0,0,0,0]}^{X=(w,w_c,c,c_1,c_2,c_3)}\big) \xrightarrow{a} \big([p_1,p_2],\ l_{\text{start}},\ [1,0,1,1,0,0]\big)$$
$$\xrightarrow{c} \big([p_1,p_4],\ l_{\text{start}},\ [10,0,2,1,0,1]\big) \xrightarrow{b} \big([p_3,p_4],\ l_{\text{start}},\ [28,0,3,1,1,1]\big)$$
$$\xrightarrow{t_5} \big([sink],\ l_{\text{start}},\ [28,0,3,1,1,1]\big) \xrightarrow{\#} \big([sink],\ l_{\text{end}},\ [28,1,3,1,1,1]\big)$$

This synchronized run generates the trace $\langle a, c, b\rangle$ and updates the LHA variables $X = (w,\ w_c,\ c,\ c_1,\ c_2,\ c_3)$ after each firing as shown above. The accumulator w evolves as follows:

- after a, $w \leftarrow 1 \cdot 3^0 = 1$ and counters become $(c, c_1) = (1, 1)$;
- after c, $w \leftarrow 1 + 3 \cdot 3^1 = 10$ with $(c, c_3) = (2, 1)$;
- after b, $w \leftarrow 10 + 2 \cdot 3^2 = 28$ with $(c, c_2) = (3, 1)$.

The silent transition t_5 brings the net to the terminal marking $[sink]$ without changing the LHA state. At this point, since the accumulated code matches the

word value $w_m(\langle a, c, b\rangle) = 28$, the corresponding autonomous transition $\#$ fires, moving the automaton to the accepting location l_{end} and setting $w_c \leftarrow 1$. This value is counted in the empirical *PDF* of w_c and contributes to estimating the sWN's stochastic language.

5.2 An ABC framework for stochastic process discovery

To discover optimal transition weights $\overline{w}$ for the sWN model S mined from an event log L over the alphabet Σ_L, we adapt the Approximate Bayesian Computation Sequential Monte Carlo (ABC–SMC) parameter inference method [13]. Our adaptation integrates the HASL-based stochastic-language estimator (Section 5.1) to evaluate, for any parameter vector $\overline{w}$, the language generated by the instantiated model $S(\overline{w})$. Concretely, the HASL component acts as a likelihood-free simulator: given $\overline{w}$, it produces synthetic executions of $S(\overline{w})$ and yields an empirical approximation $\widehat{\mathcal{S}}_{S(\overline{w})}$ of the model's stochastic language.

We compare this simulated distribution to the log distribution $\mathcal{S}_L$ using the restricted Earth Mover's Distance (rEMD), computed on the log support $\mathcal{T}_L$. The parameter space is k-dimensional, with $k = |T|$ the number of transitions of S; each vector $\overline{w} \in (0, 1]^k$ encodes an assignment of transition weights (normalised within conflict sets). Our objective is to minimise $\mathrm{rEMD}(\mathcal{S}_L, \mathcal{S}_S)$, and the resulting ABC–SMC procedure progressively concentrates particles around low-discrepancy weights (Algorithm 4). We denote by:

- n, the number of accepted particles required at each layer;
- $k = |T|$, the number of parameters (hence the particle length and the dimension of the proposal kernel);
- $W_j^{(i)}$, the normalized importance weight of particle j at level i;
- $\rho_j^{(i)} = \mathrm{rEMD}\left(\mathcal{L}_L, \widehat{\mathcal{L}}_{S(\overline{w}_j^{(i)})}\right)$, the discrepancy of particle j at level i;
- $K_i(\cdot \mid \cdot)$, the transition (proposal) kernel at level i, taken as a truncated multivariate normal random walk on $(0, 1]^k$.
- ϵ_i, the tolerance at level i, and
- $\zeta > 0$, the user-defined improvement threshold.

Given a candidate $\overline{w}$, the HASL-based procedure

$$\mathrm{CI}(S(\overline{w}),\, \varphi_{\mathrm{sle}}(L),\, \alpha,\, \delta)$$

produces an estimate $\widehat{\mathcal{L}}_{S(\overline{w})}$ of the model's stochastic language, which we compare to the log distribution $\mathcal{L}_L$ via the restricted Earth Mover's Distance (rEMD) computed on the log support. This relies on the HASL stochastic-language estimator described in Definition 5.3, applied to the parametric sWN $S(\overline{w})$ (and its synchronisation with the language detector). The estimator is controlled by the model-checker confidence level and interval width (α, δ), which govern the precision of $\widehat{\mathcal{L}}_{S(\overline{w})}$. To obtain

the multivariate normal (MVN) kernel, we use Botev's minimax tilting [16], an exponential tilt chosen by a minimax rule that yields low-variance independent and identically distributed samples from a linearly constrained multivariate normal.

Initialisation ($i = 1$). At the first level, we sample candidates from a uniform prior on the parameter space $(0, 1]^k$. Using ABC rejection with a user-defined high tolerance ϵ_1 (Alg. 1), we build the first set of accepted particles as follows:

1. *Candidate draw.* Sample a candidate $\overline{w}$ from the uniform prior on $(0, 1]^k$.
2. *HASL estimation.* Compute the estimated stochastic language

$$\widehat{\mathcal{L}}_{S(\overline{w})} \leftarrow \mathrm{CI}(S(\overline{w}),\, \varphi_{\mathrm{sle}}(L),\, \alpha,\, \delta)$$

3. *Discrepancy.* Compute the discrepancy

$$\rho \leftarrow \mathrm{rEMD}\Big(\mathcal{L}_L,\, \widehat{\mathcal{L}}_{S(\overline{w})}\Big)$$

4. *Acceptance.* Accept the candidate iff $\rho \leq \epsilon_1$; otherwise discard it and resample.
5. Repeat the above steps until n particles have been accepted, yielding the initial set of particles

$$\{\overline{w}_j^{(1)}\}_{j=1}^n$$

The initial importance weights are set to $W_j^{(1)} = 1/n$, since all accepted particles are drawn from the uniform prior and filtered only by ϵ_1. The prospective tolerance for the next level is the median discrepancy of the accepted particles:

$$\epsilon_2^{\mathrm{next}} = \mathrm{Median}\{\rho_j^{(1)}\}_{j=1}^n$$

Sequential levels ($i \geq 2$). At the beginning of level i, the tolerance is set once for all to the value computed at the previous level:

$$\epsilon_i \leftarrow \epsilon_i^{\mathrm{next}}$$

Then, for each particle $j = 1, \ldots, n$:

1. *Parent selection.* Sample an index $J \in \{1, \ldots, n\}$ with $\mathbb{P}(J = j) = W_j^{(i-1)}$. These particles will be used as the means of the truncated multivariate normal distribution for the sampling of the following particle. The covariance of this law is determined from all the accepted particles of the previous layer.
2. *Proposal.* Draw a candidate $\overline{w}^\star \sim K_i(\cdot \mid \overline{w}_J^{(i-1)})$, project it to the feasible set (clip to $(0, 1]^k$).
3. *Simulation and acceptance.* Compute $\widehat{\mathcal{L}}_{S(\overline{w}^\star)}$ by the HASL-based procedure and

$$\rho^\star \leftarrow \mathrm{rEMD}\Big(\mathcal{L}_L,\, \widehat{\mathcal{L}}_{S(\overline{w}^\star)}\Big)$$

   If $\rho^\star \leq \epsilon_i$, accept the particle and set $\overline{w}_j^{(i)} \leftarrow \overline{w}^\star$ and $\rho_j^{(i)} \leftarrow \rho^\star$; otherwise, resample/propose again.

Algorithm 4 ABC–SMC for stochastic process discovery

Require: L (event log), Σ_L (alphabet), f_l (activity mapping),
$S(\overline{w})$ (sWN structure mined from L, parameterized by $\overline{w}$),
n (number of particles), ϵ_1 (initial tolerance), ζ (improvement threshold),
$K_i(\cdot \mid \cdot)$ (transition kernel at level i),
α (confidence level), δ (interval width)

Ensure: $\{(\overline{w}_j^{(i)}, W_j^{(i)}, \rho_j^{(i)})\}_{j=1}^n$ approximating $\pi_{\mathrm{ABC},\zeta_{\epsilon_i}}$

▷ Notation: $\widehat{\mathcal{L}}_{S(\overline{w})} \leftarrow \mathrm{CI}(S(\overline{w}),\, \varphi_{\mathrm{sle}}(L),\, \alpha,\, \delta)$ is the HASL-based estimate of the model's stochastic language.

▷ Prior $\pi(\overline{w})$ is uniform on the feasible set $(0,1]^k$.

$i \leftarrow 1$

Layer $i = 1$: obtain $\{\overline{w}_j^{(1)}\}_{j=1}^n$ via ABC rejection (Alg. 1), using tolerance ϵ_1

for $j = 1$ **to** n **do**

$\quad \widehat{\mathcal{L}}_{S(\overline{w}_j^{(1)})} \leftarrow \mathrm{CI}\Big(S(\overline{w}_j^{(1)}),\, \varphi_{\mathrm{sle}}(L),\, \alpha,\, \delta\Big)$

$\quad \rho_j^{(1)} \leftarrow \mathrm{rEMD}\Big(\mathcal{L}_L,\, \widehat{\mathcal{L}}_{S(\overline{w}_j^{(1)})}\Big)$

$\quad W_j^{(1)} \leftarrow \frac{1}{n}$

end for

$\epsilon_2^{\mathrm{next}} = \mathrm{Median}\{\rho_j^{(1)}\}_{j=1}^n$

repeat

$\quad i \leftarrow i + 1$

$\quad \epsilon_i \leftarrow \epsilon_i^{\mathrm{next}}$

$\quad$ **for** $j = 1$ **to** n **do**

$\quad\quad$ **repeat**

$\quad\quad\quad$ sample $J \in \{1, \ldots, n\}$ with $\mathbb{P}(J = j) = W_j^{(i-1)}$

$\quad\quad\quad$ propose $\overline{w}^\star \sim K_i(\cdot \mid \overline{w}_J^{(i-1)})$ ▷ e.g., truncated MVN on $(0,1]^k$

$\quad\quad\quad \widehat{\mathcal{L}}_{S(\overline{w}^\star)} \leftarrow \mathrm{CI}(S(\overline{w}^\star),\, \varphi_{\mathrm{sle}}(L),\, \alpha,\, \delta)$

$\quad\quad\quad \rho^\star \leftarrow \mathrm{rEMD}\Big(\mathcal{L}_L, \widehat{\mathcal{L}}_{S(\overline{w}^\star)}\Big)$

$\quad\quad$ **until** $\rho^\star \le \epsilon_i$

$\quad\quad \overline{w}_j^{(i)} \leftarrow \overline{w}^\star$

$\quad\quad \rho_j^{(i)} \leftarrow \rho^\star$

$$\quad\quad \widetilde{W}_j^{(i)} \leftarrow \frac{\pi(\overline{w}_j^{(i)})}{\sum_{j'=1}^n W_{j'}^{(i-1)}\, K_i\Big(\overline{w}_j^{(i)} \mid \overline{w}_{j'}^{(i-1)}\Big)}$$

$\quad$ **end for**

$\quad$ normalize $W_j^{(i)} \leftarrow \widetilde{W}_j^{(i)} \Big/ \sum_{u=1}^n \widetilde{W}_u^{(i)}$ for all j

$\quad \epsilon_i^{\mathrm{next}} = \mathrm{Median}\{\rho_j^{(i)}\}_{j=1}^n$

until $\epsilon_i - \epsilon_i^{\mathrm{next}} < \zeta$

return $\big\{(\overline{w}_j^{(i)},\, W_j^{(i)}, \rho_j^{(i)})\big\}_{j=1}^n$ at tolerance ϵ_i

4. *Importance weight update.* Set the unnormalized weight

$$\widetilde{W}_j^{(i)} \leftarrow \frac{\pi(\overline{w}_j^{(i)})}{\sum_{j'=1}^{n} W_{j'}^{(i-1)} K_i\Big(\overline{w}_j^{(i)} \mid \overline{w}_{j'}^{(i-1)}\Big)}$$

where π is the prior on the feasible set (uniform in our case). At level i, the importance weight is equal to the ratio between the target and the proposal. The truncated target is

$$\overbrace{\pi_i(\overline{w})}^{\text{target}} \propto \pi(\overline{w})\,\mathbf{1}\{\rho(\overline{w}) \le \epsilon_i\} \qquad \overbrace{q_i(\overline{w})}^{\text{proposal}} = \sum_{j'=1}^{n} W_{j'}^{(i-1)} K_i\Big(\overline{w} \,\Big|\, \overline{w}_{j'}^{(i-1)}\Big)$$

Since only accepted particles are retained, the indicator cancels, yielding

$$\widetilde{W}^{(i)}(\overline{w}) \propto \frac{\pi(\overline{w})}{q_i(\overline{w})}$$

After all n particles have been accepted, the importance weights are normalised, and the prospective tolerance for the next level is set to the median discrepancy of the particles accepted at the current level:

$$\epsilon_i^{\text{next}} \leftarrow \text{Median}\big\{\rho_j^{(i)}\big\}_{j=1}^{n}$$

Stopping rule and output. The sequence of layer tolerances (ϵ_i) is non-increasing by construction. We stop when the improvement falls below the user-defined threshold ζ, i.e.,

$$\epsilon_i - \epsilon_i^{\text{next}} < \zeta$$

Upon termination, the algorithm returns the final particle set

$$\big\{(\overline{w}_j^{(i)}, W_j^{(i)}, \rho_j^{(i)})\big\}_{j=1}^{n} \quad \text{at tolerance } \epsilon_i$$

This set contains the accepted particles along with their importance weights and discrepancies, and serves as an empirical approximation of the target distribution $\pi_{\text{ABC},\zeta_{\epsilon_i}}$. Depending on the application, one may either inspect the posterior by plotting marginal histograms for each transition weight, or report a point estimate:

- the best particle $\arg\min_{1\le j\le n} \rho_j^{(i)}$ or,
- the weighted mean $\sum_{j=1}^{n} W_j^{(i)}\, \overline{w}_j^{(i)}$.

5.3 Marginal posterior distribution estimation

At the conclusion of the ABC-SMC procedure (Algorithm 4), the final population of n accepted particles $\overline{w}_1, \ldots, \overline{w}_n \in (0,1]^k$ together with their importance weights $W_1, \ldots, W_n$ provides an empirical approximation to the posterior distribution over the transition weights of the sWN. We use this population to construct, for each component of the weight vector $\overline{w}$, a marginal posterior distribution. Concretely,

for each transition $t_i \in T$, we approximate the marginal posterior of its weight using a histogram built from the final particles $\{\overline{w}_j\}_{j=1}^{n}$.

The resulting marginals enable a direct reading of model certainty. Sharp, concentrated posteriors indicate transitions whose weights are well-identified by the data, whereas broad or multimodal posteriors reveal residual uncertainty or alternative explanations within the model. Moreover, the location of the posterior mass is informative: distributions concentrated near 1 suggest that the corresponding transition must fire with high probability to match the observed behaviour. In contrast, masses near 0 indicate a limited or negligible role in generating the target language. We illustrate these interpretations on a simple real-world example in the next section.

5.3.1 Example

We report the marginal posterior distributions of the transition weights for the log `BPIC17_ol`, which captures a loan application process from a Dutch financial institution and involves eight distinct activities. Figure 5.6 shows the corresponding workflow net (WN), discovered using the Inductive Miner, together with the per-transition weight histograms. It contains eleven transitions: eight labelled with log activities and three silent (τ) transitions. For readability, activity labels are simplified as follows: *Create offer* (a), *Offer created* (b), *Offer sent mail and online* (c), *Offer sent online only* (d), *Offer returned* (e), *Offer canceled* (f), *Offer refused* (g), and *Offer accepted* (h). Because the net is acyclic (no loops allowing a transition to fire more than once), its stochastic language $\mathcal{S}_S$ is finite, which renders the estimation of transition probabilities and their marginal posteriors numerically straightforward. Moreover, it simplifies the interpretation of the histograms, clarifying the impact of each transition on the overall process.

Interpretation of posterior distributions. The complete log comprises 16 unique traces. Its stochastic language, under the simplified labelling, is:

$$\begin{aligned}\mathcal{L}_{\texttt{BPIC17\_ol}} = [\ &\langle a,b,c,f\rangle^{0.3806},\ \langle a,b,c,e,h\rangle^{0.3791},\ \langle a,b,c,e,g\rangle^{0.08215},\ \langle a,b,c,e,f\rangle^{0.05566},\\ &\langle a,b,f\rangle^{0.02798},\ \langle a,b,c,g\rangle^{0.02237},\ \langle a,b,d,e,h\rangle^{0.02161},\ \langle a,b,d,f\rangle^{0.02035},\\ &\langle a,b,d,g\rangle^{0.00235},\ \langle a,b,c\rangle^{0.00251},\ \langle a,b,c,e\rangle^{0.00112},\ \langle a,b,d,e,f\rangle^{0.00144},\\ &\langle a,b,g\rangle^{0.00137},\ \langle a,b,d,e,g\rangle^{0.000954},\ \langle a,b,d\rangle^{0.000395},\ \langle a,b,d,e\rangle^{2.33\times10^{-5}}\ ]\end{aligned}$$

All traces share the same prefix $\langle a,b\rangle$. This prefix is a pure sequence: in the sWN, a then b fire deterministically and do not belong to any conflict set. Consequently, the weights attached to a and b are *non-identifiable* from $\mathcal{L}_L$, they do not affect the sWN's stochastic language, and their marginal posteriors therefore mirror the prior, yielding (approximately) uniform histograms. This is not an issue since a and b do not belong to any conflict set; their weights can be fixed arbitrarily (e.g., set to 1) without changing the induced stochastic language. Only weights associated with genuine conflicts are informed by the data and therefore concentrate in specific regions.

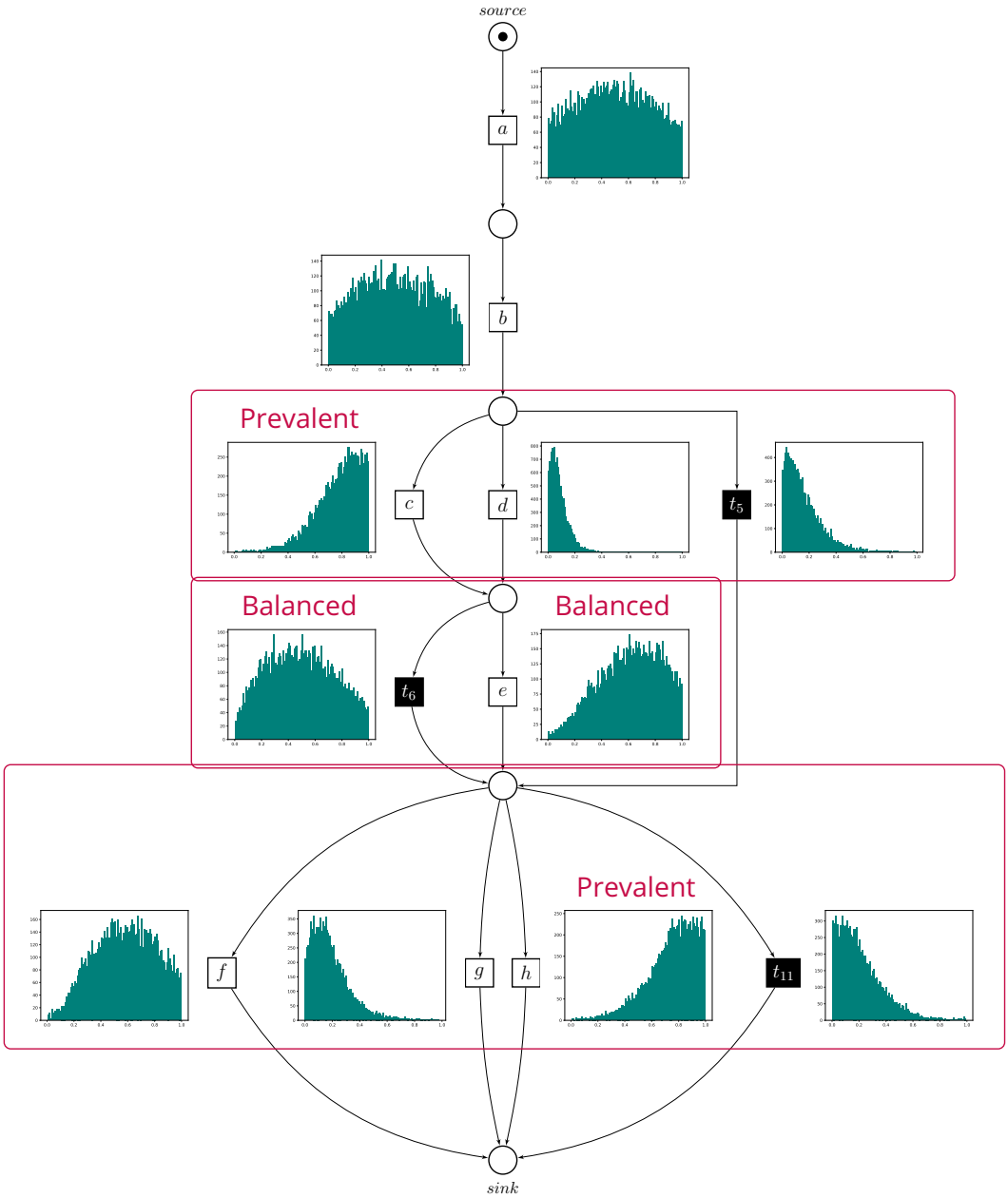


Figure 5.6 – WN discovered from the `BPIC17_o1` log

Afterwards, the process branches into a three-way conflict: fire c, fire d, or skip both. In terms of variants, seven traces take the c-branch, 7 take the d-branch, and 2 skip both. However, what matters is the *probability mass*: the c-branch concentrates about 0.9235 (92.35%), the d-branch about 0.0471 (4.71%), and skipping both about 0.0294 (2.94%). This indicates that activity c is far more likely than d or neither. In the sWN, this must be reflected in the transition weights of the corresponding conflict set: the transition labelled c should receive a much larger weight than those for d and the "skip" transition t_5. The marginal posterior histograms corroborate this pattern: they are well approximated by a truncated Gaussian-like shape on $(0, 1]$ with a narrow spread, with the mass for c concentrated near 1 and the masses for d and t_5 concentrated near 0, indicating that c must carry substantially more weight to reproduce the observed trace probabilities.

Whenever c or d occurs in a trace, activity e may follow. In the overall stochastic language, the total probability mass of traces that contain e is 0.5420 (54.20%). Restricting to traces where c or d occurs, the probability of observing e is 0.5584 (55.84%). This implies a near equiprobability between executing e and not executing it within the c/d branch. Accordingly, the marginal posterior histograms for the two competing transitions, one labelled e and its alternative t_6, exhibit broad, truncated Gaussian-like shapes on $(0, 1]$, centred near 0.56 and 0.44, respectively. The wider spread reflects *weak identifiability* of this conflict, due to the sWN firing semantics, as within a conflict set, transitions fire with probabilities proportional to their weights. Hence, the observable behaviour depends only on the *ratio* of weights, not on their absolute values: any pair (w_e, w_6) with the same ratio (e.g., $(0.2, 0.16)$ and $(0.5, 0.4)$) induces the *same* trace probabilities on the log support. This creates a ridge of equivalent solutions, yielding a broad posterior. In addition, the HASL-based language estimator has finite-sample uncertainty (controlled by (α, δ)), which further widens the histograms. Increasing the simulation budget (e.g., tightening δ at fixed α) mitigates this effect, at the expense of additional computation time.

Finally, the process terminates in a four-way conflict among f, g, h, and t_{11}. The total probability mass carried by each outcome in $\mathcal{L}_{\texttt{BPIC17\_o1}}$ is:

$$f \to 0.4861\ (48.61\%) \qquad g \to 0.1092\ (10.92\%)$$
$$h \to 0.4007\ (40.07\%) \qquad t_{11} \to 0.0040\ (0.40\%)$$

These figures indicate that f and h dominate the terminal behaviour, whereas g is comparatively rare and the "neither" outcome t_{11} is negligible. This pattern is reflected in the corresponding marginal posterior histograms.

All these results and analyses are natural and straightforward in this setting. However, the presence of more intricate control-flow patterns, such as loops or concurrency, can make the interpretation markedly more difficult: multiple parameter configurations may explain the same observed behaviour, weakening identifiability and broadening the marginal posteriors.

Posterior evolution. Returning to the `BPIC17_o1` log, the histograms shown in Figure 5.6 correspond to the results obtained after 10 layers. Figure 5.7 illustrates the

evolution of the posterior distribution constructed from the accepted weights of transition c at layers 1, 2, 5, 6, 9, and 10. The corresponding thresholds for these layers are 0.98, 0.26, 0.17, 0.16, 0.12, and 0.11, respectively. We can observe that as the threshold becomes more restrictive, the accepted weights concentrate increasingly around a specific area, resulting in a truncated Gaussian-like shape with a progressively narrower variance around the mean.

5.4 Prototype tool, experiments and results

We developed a prototype implementation of the HASL-based ABC-SMC parameter inference scheme described in Section 5.2. To validate its effectiveness, we conducted two types of experiments. The first set of experiments (Section 5.4.1) evaluates the precision and computational cost of the HASL-based stochastic language estimator introduced in Definition 5.3, considered in isolation. The second set (Section 5.4.2) assesses the ability of the ABC-SMC procedure to infer transition weights such that the stochastic language generated by the resulting sWN model closely aligns with the stochastic language observed in the event log. All experiments were performed on real-life event logs of varying complexity, with corresponding WN models discovered using the Inductive Miner algorithm [49]. This algorithm guarantees perfect fitness (equal to 1), ensuring that the discovered models are able to reproduce all traces contained in the log. The source code and experimental results are publicly available in the ProDiSt tool repository at `https://github.com/DocPierro/ProDiSt.git`. All experiments were conducted on a machine running Ubuntu equipped with a 2.60 GHz CPU.

HASL estimator engine. The prototype tool, implemented in Python, relies on the `Cosmos` model checker to compute the approximate stochastic language of a given GSPN instance. This simulation capability is integrated with a Python-based implementation of the ABC-SMC algorithm for parameter inference. `Cosmos` is a statistical model checker designed explicitly for quantitative analysis of stochastic Petri nets. Given a GSPN instance and a property expressed in HASL, `Cosmos` automatically generates a dedicated C++ simulator that synchronises the execution of the net with the automaton encoding the property. By repeatedly simulating executions, the tool collects statistical samples of the relevant random variables and computes estimators together with confidence intervals, according to user-specified precision parameters (α, δ). This approach avoids exhaustive state-space exploration, allowing for the analysis of models with very large or even infinite state spaces. In the context of our work, we exploit this capability by expressing, in HASL, the extraction of the stochastic language of a given stochastic workflow net.

5.4.1 Accuracy of HASL-based stochastic language estimation

To evaluate the accuracy of HASL-based estimation of the stochastic language of a given sWN model, we conducted a series of experiments using the `Cosmos` model

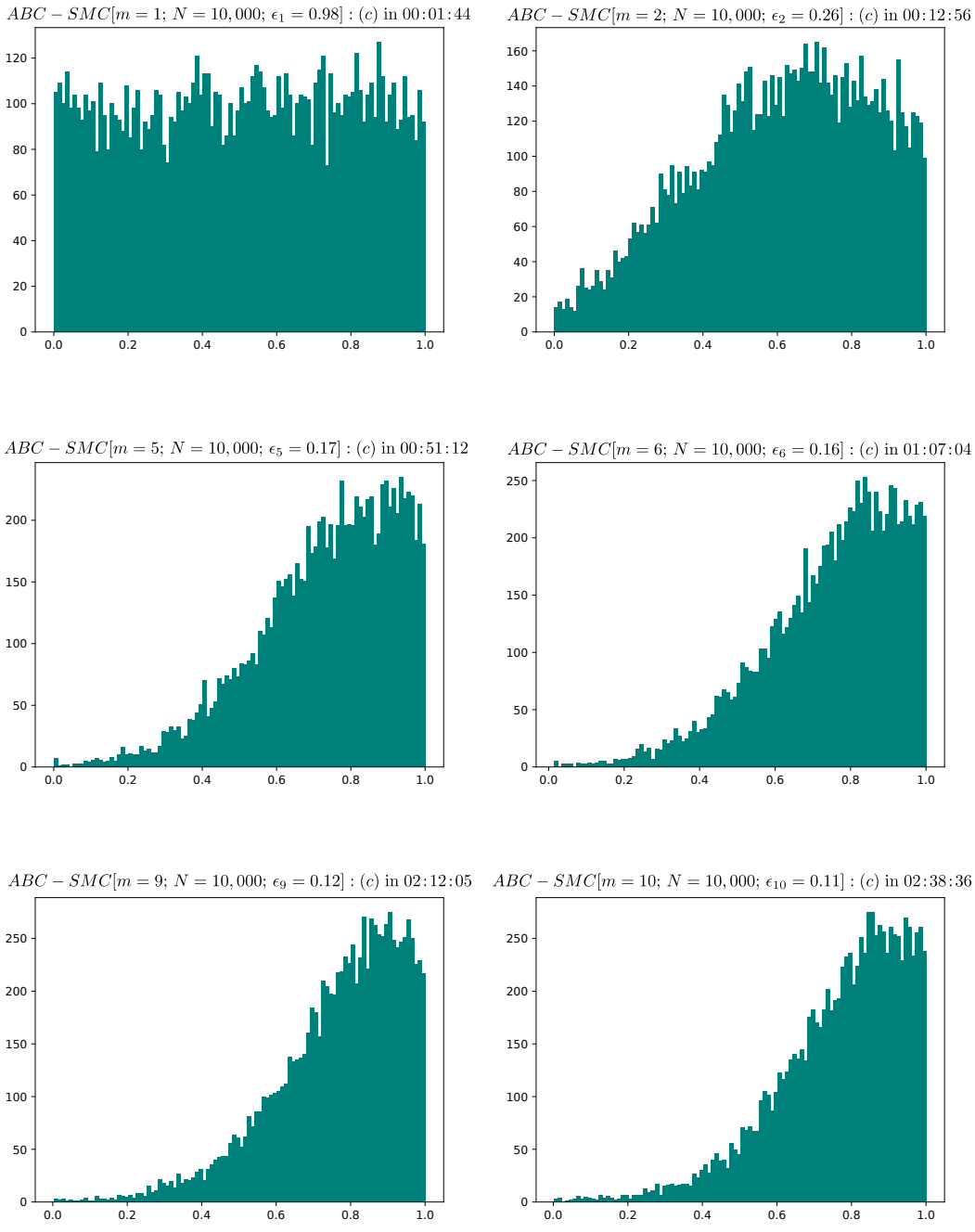


Figure 5.7 – Evolution of the posterior distribution for transition c across selected layers of the `BPIC17_o1` log

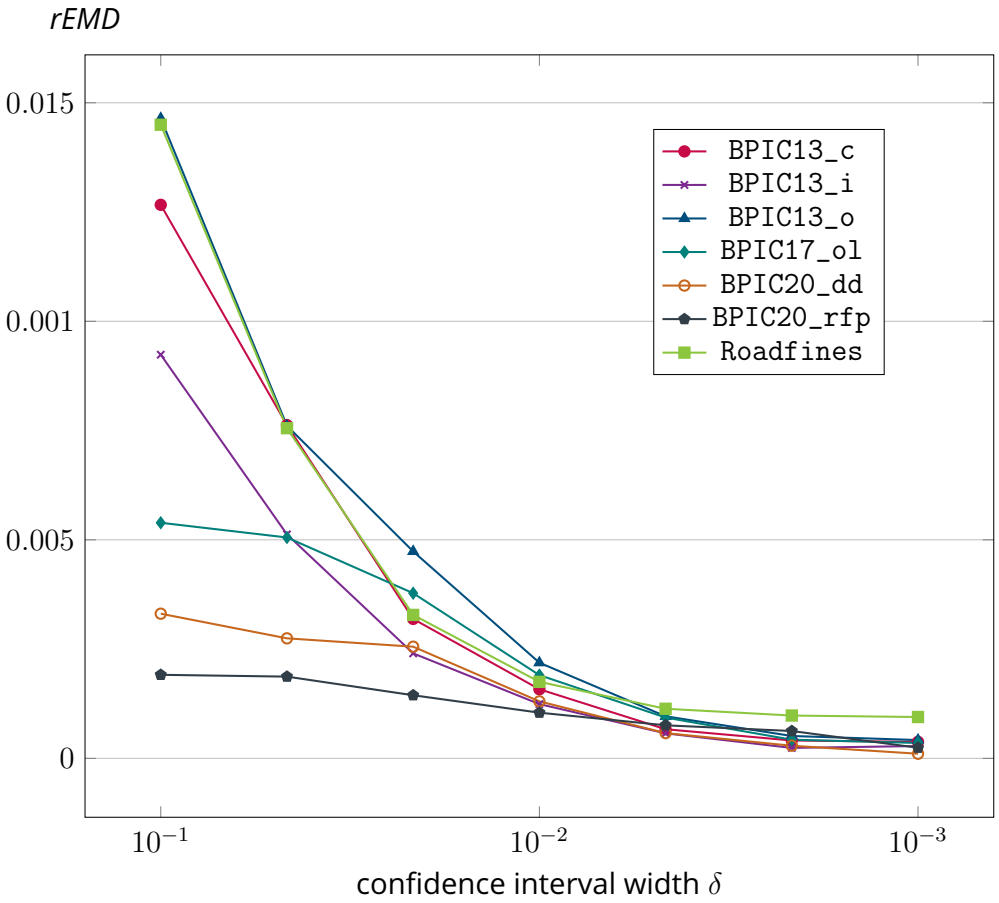


Figure 5.8 – HASL-based stochastic language approximation: quality of the approximation

checker. For each event log, we first discovered the corresponding WN N, and then generated multiple instances $S(\overline{w}_i)$ of the associated sWN by sampling random transition weight vectors $\overline{w}_i \in [0,1]^{|T|}$, where T denotes the set of transitions.

For each model instance $S(\overline{w}_i)$, we approximated its stochastic language by evaluating the HASL formula $\varphi_{sle(L)}$ on `Cosmos`, with a confidence level $\epsilon = 99\%$ and varying the confidence interval length δ. The values of ϵ and δ determine the number of simulation runs required to obtain the approximation.

We then compared the resulting approximations, in terms of rEMD distance, with the exact stochastic languages computed via the reachability graph log-driven unfolding method introduced in Chapter 3. As shown in Figure 5.8, the accuracy of the HASL-based estimation improves consistently as the confidence interval width δ decreases. Starting from $\delta = 10^{-3}$, the rEMD values become negligible for most logs (rEMD < 0.001), indicating that the estimated stochastic languages converge to the exact ones. This trend aligns with theoretical expectations, as narrower confidence intervals yield more precise statistical estimates. However, the convergence rate varies across logs, highlighting the influence of log-specific characteristics such as trace variability and event frequency.

Event log	$\delta : 10^{-1}$		$\delta : 10^{-2}$		$\delta : 10^{-3}$	
	#sim	runtime	#sim	runtime	#sim	runtime
BPIC13_c	1808	$0.21s$	114009	$1.68s$	2001301	$27.17s$
BPIC13_i	1213	$4.4s$	106793	$17.28s$	2000090	$252.91s$
BPIC13_o	1888	$0.17s$	93316	$0.96s$	2001825	$17.79s$
BPIC17_ol	10493	$0.14s$	73516	$0.31s$	2008061	$4.79s$
BPIC20_dd	2155	$0.17s$	20291	$0.32s$	1865577	$16.43s$
BPIC20_rfp	1687	$0.15s$	4984	$0.25s$	204635	$2.96s$
Roadfines	11395	$0.44s$	1071154	$19.55s$	2000746	$36.66s$

Table 5.1 – HASL-based stochastic language approximation: number of sampled traces and corresponding runtime

The table in Figure 5.8 reports the computational cost of these experiments in terms of the number of simulations and runtime for each value of δ. Combined with the rEMD plot, it illustrates the trade-off between estimation precision and computational effort. Remarkably, even for a relatively large interval width ($\delta = 10^{-1}$), the HASL-based method yields accurate approximations (rEMD < 0.015) within a few tenths of a second for all logs except one. For higher precision ($\delta = 10^{-2}$), rEMD drops below 0.001 for all logs, with runtime remaining under one second in 5 out of 7 cases. Only at the finest resolution ($\delta = 10^{-3}$) does the runtime increase significantly, confirming the feasibility of high-precision estimations.

A key challenge in applying ABC-SMC lies in the large number of particles required at each layer, which makes the procedure computationally demanding. While the HASL-based approximation provides accurate estimates of the stochastic language for a given model instance, the overall efficiency of the inference process depends critically on execution speed. In practice, a trade-off must be made. Rather than aiming for extremely high precision in each simulation, the focus is placed on obtaining sufficiently accurate estimates at a reasonable computational cost. The use of a large number of particles across successive layers compensates for the limited precision of individual evaluations. Thanks to the probabilistic and population-based nature of the ABC-SMC procedure, isolated estimation errors are averaged out and become statistically negligible in the final posterior distribution. Consequently, even moderately precise approximations of the stochastic language are sufficient to guide the inference process effectively and robustly.

5.4.2 HASL-based ABC-SMC stochastic process discovery

Table 5.2 compares the outcomes of sWN parameter estimation obtained using the ABC-SMC procedure described in Section 5.2 (column “ABC-SMC opt”) with several alternative approaches:

1. the reachability graph log-driven unfolding optimisation method from Chapter 3 (“Unfolding opt”),

Event log	$\lvert\mathcal{T}_L\rvert$	$\lvert T\rvert$	ABC-SMC opt				Unfolding opt (see Chapter 4)		Burke's estimation [21]		WaWE [17]	SLPN Miner [45]
			ζ	m	rEMD	runtime	rEMD	runtime	name	rEMD	rEMD	rEMD
BPIC13_c	183	19	0.01	12	0.082	563s	**0.04**	603s	fork	0.63	0.2	T/O
			0.005	17	0.081	1326s						
			0.0025	24	0.062	2770s						
BPIC13_i	1511	19	0.01	17	0.23	9401s	**0.18**	87086s	lhpair	0.7	0.69	T/O
			0.005	24	0.19	24427s						
			0.0025	32	**0.18**	108585s						
BPIC13_o	108	20	0.01	6	0.19	61s	**0.076**	137s	pairs	0.24	0.26	T/O
			0.005	15	0.14	340s						
			0.0025	20	0.09	1352s						
BPIC17_o1	19	11	0.01	8	0.09	197s	0.09	1.27s	freq	0.09	0.08	0.25
			0.005	19	**0.07**	434s						
			0.0025	29	**0.04**	4583s						
BPIC20_dd	99	43	0.01	23	0.086	2799s	**0.02**	3946s	fork	0.93	0.62	T/O
			0.005	28	0.082	5007s						
			0.0025	29	0.072	4583s						
BPIC20_rfp	89	51	0.01	14	0.58	2261s	0.41	59819s	freq	0.99	0.98	T/O
			0.005	28	**0.28**	7692s						
			0.0025	76	**0.08**	57634s						
Roadfines	231	34	0.01	12	0.29	1180s	0.08	1276s	pairs	0.27	0.48	T/O
			0.005	20	0.11	2312s						
			0.0025	24	**0.04**	5201s						

Table 5.2 – Comparison of rEMD distances obtained with ABC-HASL parameter inference against alternative approaches: best measured distances are in bold

2. multiple weight estimators based on statistical activity relations [21] ("Burke's estimation"),
3. the Wasserstein weight estimator [17] ("WaWE"), and
4. the stochastic labeled Petri net miner [45] ("SLPN Miner").

Column "$|\mathcal{T}_L|$" reports the number of unique traces in the log, and "$|T|$" indicates the number of transitions in the mined workflow net (i.e., the dimensionality of the parameter space). The "rEMD" columns report the restricted Earth Mover's Distance between the stochastic language of the optimised model and that of the log.

For "Burke's estimation", we considered the five estimators introduced in [21], excluding the alignment-based estimator, and retained, for each log, the one yielding the lowest rEMD value. Experiments involving the WaWE tool required the specification of five hyper-parameters that can significantly affect result quality. To ensure fairness, we adopted the reference settings provided in [17]. Following the experimental protocol of the same work, we executed each WaWE experiment 30 times, computed the rEMD for each run, and reported the median value to account for output variability. For "SLPN Miner", the optimisation procedure failed to terminate for all logs except `BPIC17_o1`. This was due to the symbolic computation of trace probabilities becoming computationally intractable in complex models. In the "ABC-SMC opt" column, ζ denotes the improvement threshold used to determine convergence (see Section 5.2), and m refers to the number of inference layers explored.

The results reported in Table 5.2 were obtained using $N = 100$ particles during ABC-SMC optimisation. The HASL-based approximation of each model's stochastic language was computed with a confidence level of $\epsilon = 99\%$ and a confidence in-

terval width $\delta = 0.1$, which, as demonstrated in Section 5.4.1, provides sufficiently accurate estimates while ensuring fast simulation times.

ABC-SMC consistently discovers more accurate parameter configurations (i.e., those yielding lower rEMD values) than "Burke's estimation", "WaWE", and "SLPN Miner". It also delivers competitive, and in some cases superior, performance compared to "Unfolding opt". While "Unfolding opt" can occasionally produce slightly better rEMD scores, ABC-SMC remains competitive, particularly when using finer improvement thresholds (e.g., $\zeta = 0.0025$), and tends to be faster for coarser thresholds (e.g., $\zeta = 0.01$). This trade-off between accuracy and efficiency makes ABC-SMC a practical choice in settings where both criteria are relevant.

The approach demonstrates robust performance across logs of varying complexity, ranging from relatively simple cases, such as `BPIC17_ol`, to large-scale scenarios, including `BPIC13_i` and `BPIC20_rfp`. For example, on `BPIC20_rfp`, which involves 51 transitions, ABC-SMC achieves a better rEMD score (0.28) than "Unfolding opt" (0.41) while requiring less than one-seventh of the runtime. Although runtime increases with log complexity, particularly for logs with many unique traces, such as `BPIC13_i`, ABC-SMC remains effective. This overhead primarily stems from the increased support of the PDF used in `Cosmos` for estimating the stochastic language, which raises the computational cost per simulation.

The particle search in ABC-SMC is inherently parallelisable, offering substantial opportunities for runtime reduction. All reported runtimes were obtained using 16 parallel jobs. On high-performance computing infrastructures, full parallelisation, matching the number of CPU cores to the number of particles per layer, could further reduce ABC-SMC runtimes, bringing them closer to those of other methods. Additional experiments varying the number of particles confirmed that reducing the particle count leads to shorter runtimes without significantly affecting rEMD values. Conversely, increasing the number of particles tends to increase runtime, with only marginal improvements in rEMD. While a larger number of particles can occasionally facilitate faster convergence, the overall accuracy gains often do not justify the additional computational cost. Nevertheless, increasing the number of particles improves the quality of the posterior distribution over transition weights. This results in a more refined estimation of parameter uncertainty and provides deeper insight into the influence of each transition on the stochastic behaviour of the net.

5.5 Conclusion

This chapter introduced a simulation-based framework for stochastic process discovery that combines HASL-based model checking with the ABC-SMC inference scheme. This approach complements the optimisation-based techniques of Chapter 4 by shifting the focus from point estimation to posterior inference, thereby capturing the uncertainty associated with transition weights in sWN. The framework relies on the `Cosmos` model checker to approximate stochastic languages through simulation, which can be integrated seamlessly within the ABC-SMC procedure.

We first validated the accuracy of the HASL-based stochastic language estimator. The experimental results confirmed that, by tuning the confidence interval parameter δ, the estimator can approximate the exact stochastic language with arbitrary precision, at the cost of increased simulation time. In practice, we observed that even moderate precision levels are sufficient for ABC-SMC to converge, since the population-based nature of the algorithm compensates for individual estimation noise.

We then assessed the performance of the complete inference framework on real-life event logs. The experiments demonstrated that the method can recover meaningful posterior distributions over transition weights, highlighting the most plausible stochastic behaviours and the degree of uncertainty in their estimation. This offers a richer interpretation than optimisation-based methods, which only provide a single best-fit parameter vector. Moreover, the results showed that the framework remains robust across logs of varying sizes and complexities, although computational costs increase with the number of particles and layers required for convergence.

Chapter 6

Stochastic Process Trees for Stochastic Process Discovery

As we explored in the previous chapter, existing approaches to stochastic process discovery mostly follow an indirect strategy. In practice, they rely on standard discovery algorithms to obtain a workflow net model from an event log, and then proceed by calibrating weight parameters for the transitions of the WN so that the resulting stochastic language resembles that of the log. While effective to some extent, this methodology presents important drawbacks. In particular, the number of parameters that need to be estimated is often unnecessarily large, and their contribution to the corresponding stochastic language is not always clear. This significantly complicates the optimisation or inference tasks required to fit the model to the observed behaviour.

To overcome these limitations, we introduce the formalism of stochastic process trees (sPTs) in this chapter. We propose to adapt the indirect discovery approach so that it operates directly on sPTs, rather than on the corresponding WN model obtained through mapping rules. The adoption of sPTs brings two significant advantages. First, the number of parameters required to describe the stochastic language is substantially reduced. Second, the role of each parameter is transparent, as it is directly tied to the control-flow operator of the process tree where it appears. This makes parameter estimation both conceptually more precise and theoretically more tractable.

We illustrate these benefits through a motivating example, and then provide a formal definition of sPTs and their stochastic semantics. We further introduce algorithms for deriving the stochastic language of an sPT, and demonstrate their applicability to the optimisation of real-life processes. The remainder of this chapter is organised as follows:

- Section 6.1 introduces a motivating example that illustrates the limitations of

WN-based stochastic discovery and the advantages of sPTs.

- Section 6.2 defines stochastic process trees and their semantics.
- Section 6.3 discuss the impact of silent loops on the stochastic semantics.
- Section 6.4 describes a simulation-based method to estimate the stochastic language of an sPT.
- Section 6.5 presents a procedure to indirectly discover an sPT via optimisation.
- Section 6.6 concludes the chapter.

6.1 A motivating example

Let us consider a widely used, publicly available event log, namely `BPIC13_o`, which records observed executions of an incident management system at Volvo IT. This log is part of the benchmark dataset collection used throughout this thesis. The traces are built on an alphabet of three actions: *Accepted* (a), *Queued* (q) and *Completed* (c), which, respectively, refer to the acceptance, the queuing and the completion of an incident query. Figure 6.1 illustrates the process tree discovered with the Inductive Miner algorithm from the log, while Figure 6.2 presents its translation into a workflow net.

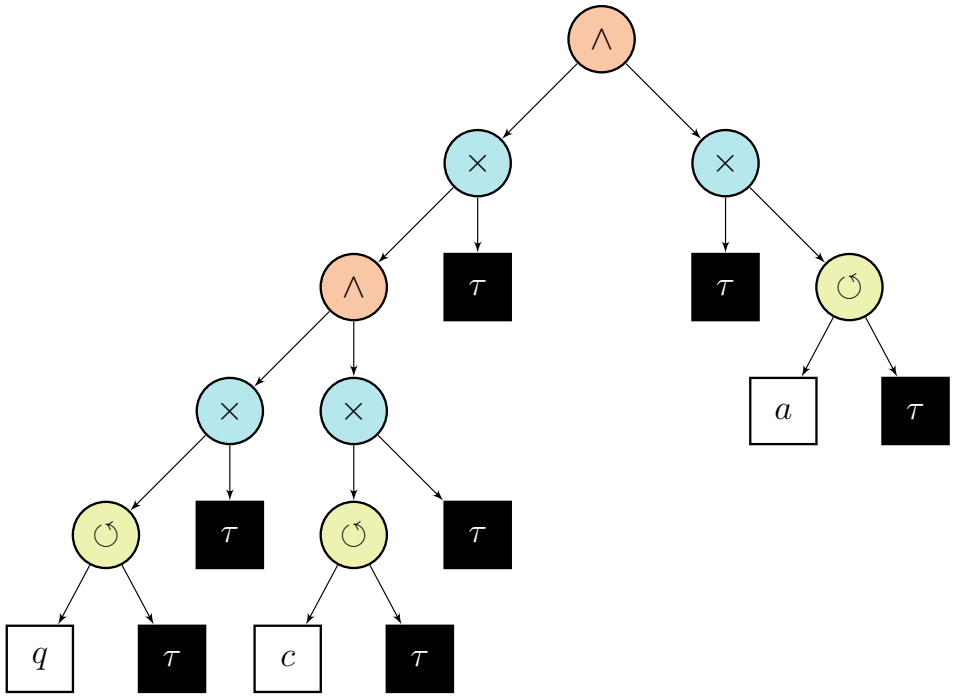


Figure 6.1 – The PT for the `BPIC13_open` log obtained through the Inductive Miner algorithm

Looking at the block structure of the WN in Figure 6.2, we observe that it consists of two main parallel components. The bottom component contains an optional a-loop block, while the top component further combines two parallel blocks

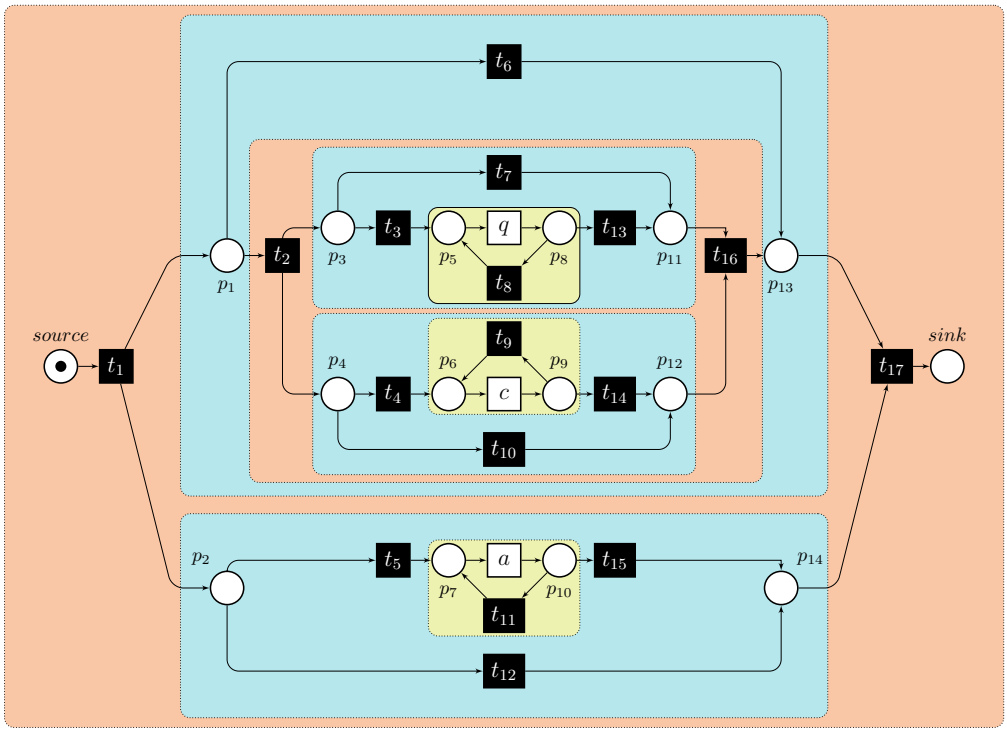


Figure 6.2 – The WN for the `BPIC13_open` log obtained through the Inductive Miner algorithm

containing an optional q-loop and an optional c-loop, respectively. As a result, the net is capable of generating traces that may contain an arbitrary number of a actions (issued by the a-loop in the bottom component), interleaved with an arbitrary number of shuffled q and c actions (issued by the two loops in the top component). Observe that the WN consists of 20 transitions, 17 of which are silent, while the remaining three correspond to the activities a, c, and q. In the context of indirect stochastic process discovery, this implies that an algorithm must determine the optimal values for all 20 weight parameters, taking into account how each transition weight influences the probability of every trace in the stochastic language generated by the resulting sWN. Recall that the probability of a trace (i.e., a trace such as $\langle a, q, q, q, c\rangle$ or $\langle a, c, a, a, q\rangle$) is obtained by summing the probabilities of all transition firing sequences that yield that trace. The difficulty, in this respect, arises from the likely presence of parallel blocks (modelling concurrency) and the potentially large number of silent transitions in the discovered WN. As a consequence, many distinct firing sequences may correspond to the same trace. From a parameter optimisation perspective, this significantly increases complexity, since the weights of numerous silent transitions must be considered even in situations where, due to the absence of *structural causality*, their values should, in principle, not influence the probability of the resulting trace.

Let us illustrate this phenomenon using the WN discovered from the `BPIC13_o` log. Consider the sequence $\sigma_1 = \langle a, q, c\rangle$. Despite its extreme simplicity, this sequence can be produced by 604 firing sequences of the net. We count here the number of complete firing sequences (including silent transitions) whose visible

projection is precisely the word $\langle a, q, c\rangle$ and that terminate in marking $[sink]$. To prevent extra visible labels, we do not use the local cycles through t_8, t_9, or t_{11} (each would enable an additional q, c, or a, respectively). Likewise, we avoid detours that would remove the possibility of producing the required labels (e.g., t_6 or t_{12}). The set of used transitions is therefore:

$$\{t_1,\, t_2,\, t_3,\, q,\, t_{13},\, t_4,\, c,\, t_{14},\, t_5,\, a,\, t_{15},\, t_{16},\, t_{17}\}.$$

The precedence constraints induced by the net arcs are:

$$\begin{aligned}
&t_1 \prec t_2, \quad t_1 \prec t_5,\\
&t_2 \prec t_3, \quad t_2 \prec t_4,\\
&t_3 \prec q \prec t_{13}, \quad t_4 \prec c \prec t_{14}, t_5 \prec a \prec t_{15},\\
&\{t_{13}, t_{14}\} \prec t_{16}, \quad \{t_{16}, t_{15}\} \prec t_{17},
\end{aligned}$$

and the visible-order constraint imposed by the target trace is:

$$a \prec q \prec c.$$

We must therefore count the linear extensions of this partial order to obtain the 604 different combinations.

Even for such a short trace, the number of alternative firing sequences is significant, and each involves a non-trivial propagation of tokens through silent transitions. Notice that the silent transition t_{11} (responsible for re-activating transition a, and thus enabling multiple occurrences of a in a trace) is enabled as long as p_{10} is marked. Along any execution realising the visible trace $\langle a, q, c\rangle$, this occurs exactly in the following reachable markings:

$$\begin{aligned}
\{&[p_1, p_{10}],\ [p_3, p_4, p_{10}],\ [p_4, p_5, p_{10}],\ [p_3, p_6, p_{10}],\ [p_5, p_6, p_{10}],\\
&[p_4, p_8, p_{10}],\ [p_6, p_8, p_{10}],\ [p_8, p_9, p_{10}],\ [p_8, p_{12}, p_{10}],\\
&[p_{11}, p_4, p_{10}],\ [p_{11}, p_6, p_{10}],\ [p_{11}, p_9, p_{10}],\ [p_{11}, p_{12}, p_{10}],\ [p_{10}, p_{13}]\}
\end{aligned}$$

This implies that the weight of t_{11} influences the probability distribution over how activities q and c are executed. Moreover, this influence depends in multiple ways on the weights of the transitions with which t_{11} is concurrently enabled. From a logical standpoint, this is problematic: transitions q and c belong to blocks that are not causally connected to t_{11}. Hence, the semantics of the sWN are responsible for such an illogical situation that happens every time we have concurrency within more complex patterns. A surprisingly large number of such situations already arises in a relatively small model.

On the contrary, let us consider the PT of the `BPIC13_o` log in Figure 6.1, and imagine that each arc connecting a node to one of its children is annotated with a probability parameter p_i (the parameters of the arcs outgoing from a given node must sum up to 1). In this case, there are nine probability parameters, p_i with $1 \leq i \leq 9$, which suffice to associate a probability value with each trace that the sPT can generate (the formal semantics will be introduced in Section 6.2). Besides the considerable reduction in the number of parameters (9 in the PT of Figure 6.2

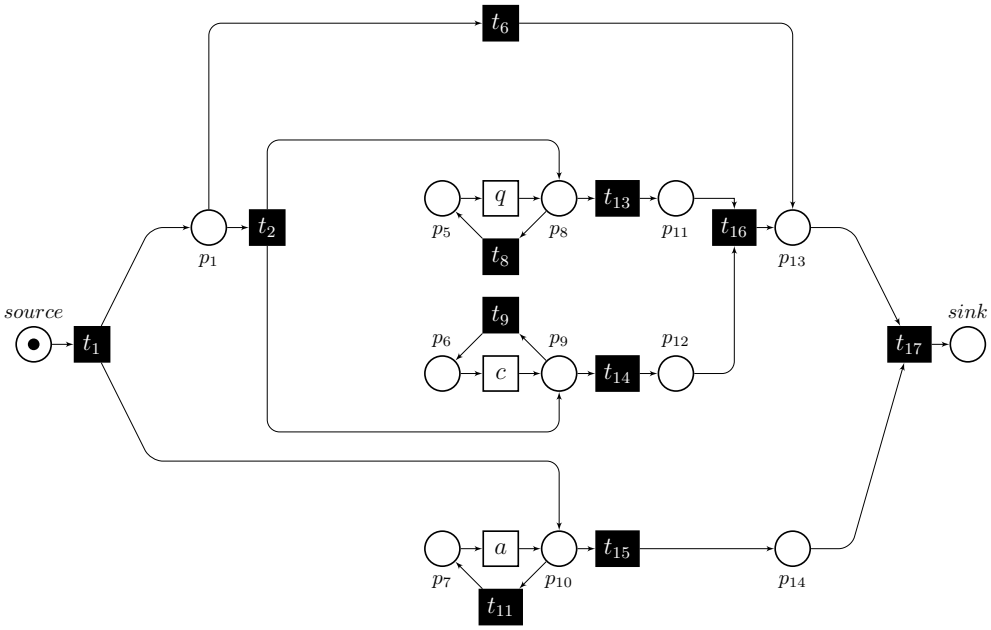


Figure 6.3 – A semantically equivalent WN to that of Figure 6.2

versus 20 in the corresponding WN of Figure 6.2), which already simplifies parameter optimisation, the probability-enriched PT model brings an additional advantage: it prevents the probability of the traces generated by the model from being influenced by parameters of transitions that are causally unrelated (as discussed above). In other words, within the stochastic process tree formalism introduced in Section 6.2, the role of each probability parameter is strictly *local* to the sub-tree rooted at its operator. As a result, parameters have a clear and well-defined impact on the stochastic language generated by the process tree.

Another drawback of introducing stochasticity at the WN level, rather than at the PT level, is the lack of univocality in the discovered WN. The Inductive Miner algorithm first extracts a PT from a log and then derives a corresponding WN through mapping rules. We stress, however, that the resulting WN is not necessarily the unique representation of the PT extracted by Inductive Miner. It is straightforward to show that several semantically equivalent WN models can be obtained, for instance, by modifying the structure produced by the mapping rules. By *semantically equivalent*, we mean that both WNs generate a language with the same support.

For example, Figure 6.3 shows an alternative workflow net obtained from the Inductive Miner mapping in Figure 6.2 by removing a few transitions and places. Specifically, we remove

- the transitions that enable entry into the loops component $\{t_3, t_4, t_5\}$,
- and the skip transitions for the loops pattern $\{t_7, t_{10}, t_{12}\}$.

We preserve the same semantics by directly connecting the transition that initiates the loop to the place that enables an immediate exit from the loop, without firing the loop body. It is straightforward to verify that the two nets are language-

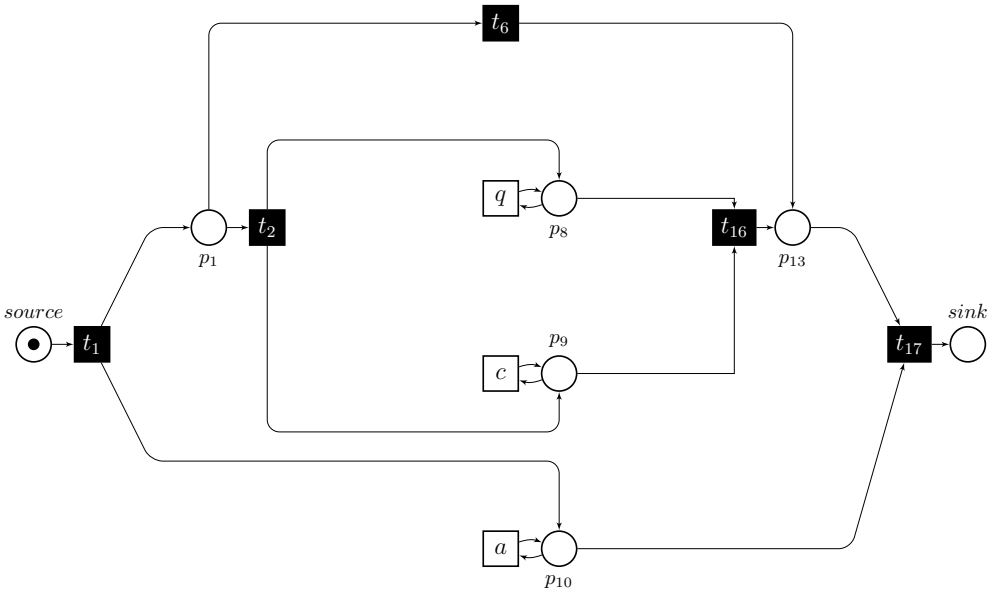


Figure 6.4 – Another semantically equivalent WN to that of Figure 6.2

equivalent (i.e., they generate the same set of traces). However, the net in Figure 6.3 is clearly less complex and contains substantially fewer transitions (11 instead of 17). A stronger simplification is depicted in Figure 6.4, which now contains only 8 transitions. We delete the loop-condition silent transitions t_8, t_9, and t_{11}, and we contract the exit-confirmation τ-transitions t_{13}, t_{14}, and t_{15} by wiring the labeled transitions (q, c, a) directly to their post-places (p_8, p_9, p_{10}), which already feed the concurrency exits t_{16} and t_{17}. Because these edges are silent, this reduction preserves the visible language. The simplified net is therefore equivalent to the original, yet substantially smaller in size.

As a consequence, we stress that several different WNs can be obtained, each with a different number of parameters, and these will inevitably behave differently when it comes to parameter optimisation. By introducing the stochastic extension of the process tree formalism, we avoid this inconvenience: the optimisation step can operate directly at the process tree level, rather than on non-unique WN models.

For the sake of completeness, we also mention [26], where the authors investigate the role of weights associated with immediate transitions in Petri nets combining timed and immediate transitions. They suggest that applying true-concurrency semantics to *non-conflicting* transitions may, in some cases, reduce the number of parameters. Although potentially relevant, these findings do not directly apply to our setting. When determining the (visible) language of a WN, one must account for the distinction between labeled and silent (τ) transitions and for the fact that, even for non-conflicting transitions, the firing order can affect the language whenever both transitions are labeled (the only benign case is concurrently enabled, non-conflicting *silent* transitions, whose order does not change the visible projection). Moreover, mined WNs often exhibit markings in which transitions from different

effective conflict sets are concurrently enabled, further limiting the applicability of those reductions.

6.2 Definition and semantics

A process tree is associated with a language, that is, the set of traces it can generate. To capture stochastic languages, where each trace is not only possible but also assigned a probability, we extend process trees with probabilistic mechanisms. In this section, we first introduce the formal definition of *stochastic process trees* (sPT). We then provide intuitive explanations of the stochastic operators through illustrative examples, before presenting the formal semantics of sPTs.

Definition 6.1 (Stochastic Process Tree) Let Σ_Q be a finite alphabet of process activities and $\mathcal{O} = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ the set of process tree operators. The set of stochastic process trees (sPTs) over Σ_Q is defined inductively as follows:

- **Leaf.** If $a \in \Sigma_Q \cup \{\tau\}$, then $Q = a$ is an sPT.
- **Sequence.** If $Q_1, \ldots, Q_n$ with $n \geq 2$ are sPTs, then

$$Q = \rightarrow (Q_1, Q_2, \ldots, Q_n)$$

- **Probabilistic choice / parallel.** If $Q_1, \ldots, Q_n$ with $n \geq 2$ are sPTs, and $(p_1, \ldots, p_n)$ is a probability distribution with $p_i \geq 0$ and $\sum_{i=1}^{n} p_i = 1$, then for $\oplus \in \{\times, \wedge\}$

$$Q = \oplus(Q_1, \ldots, Q_n, p_1, \ldots, p_n)$$

- **Probabilistic loop.** If Q_1 and Q_2 are sPTs and $p \in [0, 1]$, then

$$Q = \circlearrowleft (Q_1, Q_2, p)$$

Before presenting the formal semantics of sPTs, we first provide an intuitive explanation supported by simple examples. In these examples, the language of a tree Q_i is denoted by $\mathcal{S}_{Q_i}$.

Leaf. Leaves represent the atomic steps of a process. They can be labelled with either a log activity $a \in \Sigma_Q$ or with the silent activity τ. The stochastic language of a leaf always consists of a single trace: the empty trace ε when the label is τ, and otherwise the one-activity trace. In both cases, this unique trace has probability 1.

Figure 6.5 – Q_1: tree with a single activity labelled leaf

Figure 6.6 – Q_2: tree with a single τ labelled leaf

The stochastic language of Q_1 in Figure 6.5 is $\mathcal{S}_{Q_1} = [\langle a \rangle^1]$, while that of Q_2 in Figure 6.6 is $\mathcal{S}_{Q_2} = [\varepsilon^1]$.

Choice. The choice operator, denoted by $\times$, models a decision among multiple children where exactly one child is selected for execution. Each child is assigned a probability, and the probabilities must sum to one. Consequently, for a choice node with n children, the number of independent parameters is $n - 1$. The semantics is as follows: first, one child is selected according to the given probabilities; then, a trace is drawn from the stochastic language of the selected child.

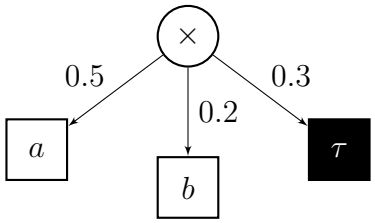


Figure 6.7 – Q_3: a tree whose root is a choice operator.

Q_3 in Figure 6.7 represents a choice among three leaves labelled by a, b, and τ. Each child yields a single-trace language, and the resulting stochastic language of the tree is:

$$\mathcal{S}_{Q_3} = [\langle a \rangle^{0.5},\ \langle b \rangle^{0.2},\ \varepsilon^{0.3}]$$

Sequence. The sequence operator, denoted by $\rightarrow$, enforces a strict left-to-right execution order of its children. From each child, a trace is sampled according to its stochastic language, and the resulting traces are concatenated.

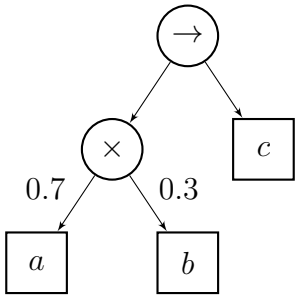


Figure 6.8 – Q_4: a tree whose root is a sequence operator

Q_4 in Figure 6.8 represents a choice between activities a and b, followed, due to the sequence operator, by activity c. Thus, every trace begins with either a (with probability 0.7) or b (with probability 0.3), and ends with c. The resulting stochastic language is:

$$\mathcal{S}_{Q_4} = [\langle a, c \rangle^{0.7},\ \langle b, c \rangle^{0.3}]$$

Parallel. The parallel operator, denoted by $\wedge$, allows its children to be executed concurrently in an interleaved manner. Each child is assigned a probability, and the probabilities must sum to one. For a parallel node with n children, this results in $n - 1$ independent parameters. The semantics is as follows: first, a trace is drawn from the stochastic language of each child; then, the resulting traces are interleaved, where the choice of the next action is guided by the probabilities assigned to the children.

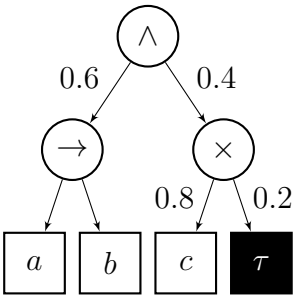


Figure 6.9 – Q_5: a tree whose root is a parallel operator

Figure 6.9 shows a parallel between a sequence node generating $[\langle a, b\rangle^1]$ and a choice node generating $[\langle c\rangle^{0.8},\ \epsilon^{0.2}]$. The parallel operator computes all interleavings between the traces of its children.

Case 1: $\langle a, b\rangle$ is interleaved with $\langle c\rangle$. Here, three distinct interleavings are possible:

- $\langle c, a, b\rangle$: The probability of selecting c as the first element of the interleaving is 0.4. Once c is placed, the second child is exhausted, and the remainder of the interleaving must follow the trace $\langle a, b\rangle$ from the first child. Since this continuation is deterministic, the final probability of obtaining $\langle c, a, b\rangle$ is $0.8 \cdot 0.4 = 0.32$.
- $\langle a, c, b\rangle$: The interleaving begins with a (probability 0.6), followed by c from the second child (probability 0.4), and finally b. The overall probability of $\langle a, c, b\rangle$ is $0.8 \cdot 0.6 \cdot 0.4 = 0.192$.
- $\langle a, b, c\rangle$: this case occurs when a and b from the first child are taken before c from the second child. The trace $\langle c\rangle$ is selected with probability 0.8, while a and b are chosen first with probability 0.6 each. Hence, the overall probability is $0.8 \cdot 0.6 \cdot 0.6 = 0.288$.

Case 2: $\langle a, b\rangle$ is interleaved with ε. Since the second child produces no visible activity, the only resulting trace is $\langle a, b\rangle$. However, the interleaving procedure may still place the empty contribution at different positions, leading to three distinct interleavings:

- τ first, then a, then b with probability $0.2 \cdot 0.4 = 0.08$,
- a, then τ, then b with probability $0.6 \cdot 0.2 \cdot 0.4 = 0.048$,
- a, b, then τ with probability $0.6 \cdot 0.6 \cdot 0.2 = 0.072$.

Summing these possibilities yields:

$$\mathcal{S}_{Q_5}(\langle a, b\rangle) = 0.08 + 0.048 + 0.072 = 0.2$$

Resulting stochastic language. Combining both cases yields:

$$\mathcal{S}_{Q_5} = [\langle c, a, b\rangle^{0.32},\ \langle a, c, b\rangle^{0.192},\ \langle a, b, c\rangle^{0.288},\ \langle a, b\rangle^{0.2}]$$

Loop. The loop operator, denoted by $\circlearrowleft$, has exactly two children. The left child represents the *do* part, which is executed at least once, while the right child represents the *redo* part, executed before each repetition of the *do* part. The parameter p is the probability of repeating the loop; consequently, the loop terminates with probability $1 - p$. If $p > 0$ and the body contains non-silent actions, the loop generates an infinite stochastic language.

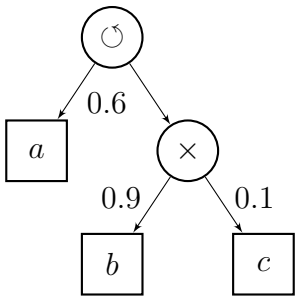


Figure 6.10 – Q_6: a tree whose root is a loop operator

In Q_6 (Figure 6.10), the left child is a leaf labelled a with stochastic language $\mathcal{S}_a = [\langle a \rangle^1]$, while the right child is a choice between b and c, with $\mathcal{S}_\times = [\langle b \rangle^{0.9},\ \langle c \rangle^{0.1}]$. The repetition probability is $p = 0.6$. The probabilities of the first few traces are:

$$
\begin{aligned}
\text{zero loop: } & \mathcal{S}_{Q_6}(\langle a \rangle) = 1 \cdot (1-p) = 0.4, \\
\text{one loop: } & \mathcal{S}_{Q_6}(\langle a, b, a \rangle) = 1 \cdot p \cdot 0.9 \cdot 1 \cdot (1-p) = 0.216, \\
& \mathcal{S}_{Q_6}(\langle a, c, a \rangle) = 1 \cdot p \cdot 0.1 \cdot 1 \cdot (1-p) = 0.024, \\
\text{two loops: } & \mathcal{S}_{Q_6}(\langle a, b, a, b, a \rangle) = 1^3 \cdot 0.9^2 \cdot p^2 \cdot (1-p) = 0.11664, \\
& \mathcal{S}_{Q_6}(\langle a, c, a, c, a \rangle) = 1^3 \cdot 0.1^2 \cdot p^2 \cdot (1-p) = 0.00144, \\
& \mathcal{S}_{Q_6}(\langle a, b, a, c, a \rangle) = \mathcal{S}_{Q_6}(\langle a, c, a, b, a \rangle) \\
& \qquad = 1^3 \cdot 0.9 \cdot 0.1 \cdot p^2 \cdot (1-p) = 0.01296, \\
\text{three loops: } & \ldots
\end{aligned}
$$

To formally define the semantics of sPTs, and in particular the parallel operator which relies on interleaving, we first introduce a *stochastic shuffler* function.

Definition 6.2 (Stochastic Shuffle) Let $\sigma_1, \ldots, \sigma_n$ be traces over alphabet Σ, and let $(p_1, \ldots, p_n)$ be a probability distribution with $\sum_{i=1}^{n} p_i = 1$. We denoted by:

$$P(Sh(\sigma_1, \ldots, \sigma_n, p_1, \ldots, p_n) = \sigma)$$

the probability that the stochastic shuffle of traces $\sigma_1, \ldots, \sigma_n$ with probabilities $p_1, \ldots, p_n$ yields the trace σ. A shuffle can be described by a sequence of indices

$$s = [i_1, i_2, \ldots, i_{|\sigma|}], \qquad 1 \leq i_j \leq n \ \ (1 \leq j \leq |\sigma|)$$

where i_j indicates that the j-th activity of σ is taken from trace σ_i. Given such a sequence s, we write $R(\sigma_1, \ldots, \sigma_n, s)$ for the trace obtained by interleaving $\sigma_1, \ldots, \sigma_n$ according to the order prescribed by s.

Given a shuffling $s = [i_1, \ldots, i_{|\sigma|}]$, we denote by $\mathcal{C}(s,k)$ the set of indices of traces that still contribute activity after position k, i.e.,

$$\mathcal{C}(s,k) = \{\, i_j \mid j > k \,\}$$

Further, let $S(\sigma_1, \ldots, \sigma_n, \sigma)$ denote the set of all shufflings s of traces $\sigma_1, \ldots, \sigma_n$ that result in σ, i.e.,

$$\mathcal{S}(\sigma_1, \ldots, \sigma_n, \sigma) = \{\, s \mid R(\sigma_1, \ldots, \sigma_n, s) = \sigma \,\}$$

With this notation, the probability that the stochastic shuffle of $\sigma_1, \ldots, \sigma_n$ with probabilities $p_1, \ldots, p_n$ yields σ is

$$\mathbb{P}(Sh(\sigma_1, \ldots, \sigma_n, p_1, \ldots, p_n) = \sigma) = \sum_{s \in S(\sigma_1,\ldots,\sigma_n,\sigma)} \prod_{k=1}^{|\sigma|} \frac{p_{s[k]}}{\sum_{i \in \mathcal{C}(s,k)} p_i} \tag{6.1}$$

where the product represents the probability that the shuffle proceeds exactly according to s.

Q Example 6.1 (Stochastic shuffles of two traces).

Let $\sigma_1 = \langle a, a, b \rangle$ and $\sigma_2 = \langle c, d \rangle$ be two traces over alphabet $\Sigma = \{a, b, c\}$, and $s = [1, 1, 2, 2, 1]$, $s' = [2, 2, 1, 1, 1]$ two shuffling sequences referred to σ_1 and σ_2. A shuffling s indicates the order in which elements of the shuffled traces should be selected while constructing the corresponding interleaving. Therefore $s = [1, 1, 2, 2, 1]$ indicates that the interleaving shall be obtained by first selecting an element of σ_1 (as $s[1] = 1$), then appending to it the next element of σ_1 (as $s[2] = 1$), then appending an element of σ_2 (as $s[3] = 2$), and so on, where $s' = [2, 2, 1, 1, 1]$ indicates that the first two elements of the shuffles shall be picked from σ_2 ($s'[1] = s'[2] = 2$) while the remaining three (obviously) from σ_1 ($s'[3] = s'[4] = s'[5] = 1$). The resulting interleaving of σ_1 and σ_2 according to shuffle s and s' are:

$$R(\sigma_1, \sigma_2, s) = R(\langle a, a, b \rangle, \langle c, d \rangle, [1, 1, 2, 2, 1]) = \langle a, a, c, d, b \rangle$$
$$R(\sigma_1, \sigma_2, s') = R(\langle a, a, b \rangle, \langle c, d \rangle, [2, 2, 1, 1, 1]) = \langle c, d, a, a, b \rangle$$

Given a shuffle sequence s, $\mathcal{C}(s, i) \subseteq \{1, 2\}$ (with $i \in \{1, |s|\}$) indicates which, amongst the two traces being shuffled, can still contribute to the construction of the interleaving when the i-th element is yet to be added to the interleaved trace. Therefore w.r.t. shuffling $s = [1, 1, 2, 2, 1]$ and $s' = [2, 2, 1, 1, 1]$ of σ_1 and σ_2 we have:

$$\mathcal{C}(s,1) = \mathcal{C}(s,2) = \mathcal{C}(s,3) = \mathcal{C}(s,4) = \{1,2\} \quad \text{and} \quad \mathcal{C}(s,5) = \{1\}$$
$$\mathcal{C}(s',1) = \mathcal{C}(s',2) = \{1,2\} \quad \text{and} \quad \mathcal{C}(s',3) = \mathcal{C}(s',4) = \mathcal{C}(s',5) = \{1\}$$

If we now consider the following shuffle probabilities $p_1 = \frac{1}{3}$ and $p_2 = \frac{2}{3}$ we can compute, by application of 6.1), the probability of the the interleaving of σ_1 and σ_2 resulting from shuffles $s = [1, 1, 2, 2, 1]$ and $s' = [2, 2, 1, 1, 1]$ as follows:

$$\mathbb{P}(Sh(\sigma_1, \sigma_2, p_1, p_2) = \langle a, a, c, d, b \rangle) = \frac{\frac{1}{3}}{\frac{1}{3}+\frac{2}{3}} \cdot \frac{\frac{1}{3}}{\frac{1}{3}+\frac{2}{3}} \cdot \frac{\frac{2}{3}}{\frac{1}{3}+\frac{2}{3}} \cdot \frac{\frac{2}{3}}{\frac{1}{3}+\frac{2}{3}} \cdot \frac{\frac{1}{3}}{\frac{1}{3}} = \frac{4}{81}$$

$$\mathbb{P}(Sh(\sigma_1, \sigma_2; p_1, p_2) = \langle c, d, a, a, b \rangle) = \frac{2}{3} \cdot \frac{2}{3} \cdot 1 \cdot 1 \cdot 1 = \frac{4}{9}$$

Notice that at step k, as long as both traces still have activities ($\mathcal{C}(s, k) = \{1, 2\}$), the denominator is $p_1 + p_2 = 1$, yielding factors $\frac{1}{3}$ or $\frac{2}{3}$. Once only one trace remains active, the denominator equals its p_i, resulting in a factor of 1.

Definition 6.3 (Semantics of sPTs) Let Q be an sPT. The stochastic language $\mathcal{S}_Q$ of Q is defined recursively as follows:

- if $Q = a$ with $a \in \Sigma_Q$, then $\mathcal{S}_Q(\langle a \rangle) = 1$;
- if $Q = \tau$, then $\mathcal{S}_Q(\varepsilon) = 1$;
- (sequence:) if $Q = \rightarrow (Q_1, \ldots, Q_n)$, then

$$\mathcal{S}_Q(\sigma) = \sum_{\sigma_1. \ \ldots \ .\sigma_n = \sigma} \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i);$$

- (choice:) if $Q = \times(Q_1, \ldots, Q_n; p_1, \ldots, p_n)$, then

$$\mathcal{S}_Q(\sigma) = \sum_{i=1}^{n} p_i \mathcal{S}_{Q_i}(\sigma)$$

- (parallel:) if $Q = \wedge(Q_1, \ldots, Q_n; p_1, \ldots, p_n)$, then

$$\mathcal{S}_Q(\sigma) = \sum_{\sigma_1, \sigma_2, \ldots, \sigma_n} \left(\mathbb{P}(Sh(\sigma_1, \ldots, \sigma_n; p_1, \ldots, p_n) = \sigma) \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i) \right);$$

- (loop:) if $Q = \circlearrowleft (Q_1, Q_2; p)$, then

$$\mathcal{S}_Q(\sigma) = \sum_{\substack{m \in \mathbb{N} \\ \sigma_1 . \sigma'_1 . \ \ldots \ . \sigma'_{m-1} . \sigma_m = \sigma}} \left(p^{(m-1)} (1-p) \prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \prod_{i=1}^{m-1} \mathcal{S}_{Q_2}(\sigma'_i) \right).$$

The parallel operator is defined with one parameter per component and is used uniformly throughout the shuffling. This entails the implicit assumption that the choice of the following action is independent of the current state, i.e., of the portion of the parallel block that has already been executed. Although this assumption could be relaxed, doing so would require a substantially more complex parametrisation of the model.

The loop operator is defined so that the number of executions follows a geometric distribution. This choice ensures that the underlying stochastic process is memoryless, which facilitates efficient non-simulation-based analysis. Suppose empirical evidence from event logs indicates that the number of repetitions of an activity (or a sequence of activities) is not well captured by a geometric distribution. In that case, the formalism can be naturally extended to support alternative distributions.

Theorem 6.1 An sPT Q, satisfying the syntax in Definition 6.1 and interpreted according to the semantics in Definition 6.3 is associated with a normalised stochastic language.

Proof. We write $\mathcal{S}_Q : \Sigma^* \to [0,1]$ for the stochastic language induced by Q (as in Definition 6.3). By structural induction on Q we prove that

$$\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) = 1.$$

Leaf. Let Q be a leaf labelled either by $a \in \Sigma_Q$ or by τ. By the semantics, we have a unit mass on the one-letter trace $\sigma = \langle a \rangle$ or on the empty trace $\sigma = \varepsilon$

$$\mathcal{S}_Q = [\langle a \rangle^1] \quad \text{and} \quad \mathcal{S}_Q = [\varepsilon^1].$$

In both cases, we have

$$\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) = 1.$$

Sequence. Let $Q = \to (Q_1, \ldots, Q_n)$ with $n \geq 2$. By induction on n, let us prove

$$P(n) : \qquad \sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) = \sum_{\sigma \in \Sigma^*} \sum_{\sigma_1 . \sigma_2 . \ldots . \sigma_n = \sigma} \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i) = 1.$$

- *Base* $n = 2$: By the semantics of sequence,

$$\begin{aligned}
\sum_{\sigma \in \Sigma^*} \mathcal{S}_{\to(Q_1,Q_2)}(\sigma) &= \sum_{\sigma \in \Sigma^*} \sum_{u \cdot v = \sigma} \mathcal{S}_{Q_1}(u) \mathcal{S}_{Q_2}(v) \\
&= \sum_{u \in \Sigma^*} \sum_{v \in \Sigma^*} \mathcal{S}_{Q_1}(u) \mathcal{S}_{Q_2}(v) \\
&= \sum_{u \in \Sigma^*} \mathcal{S}_{Q_1}(u) \sum_{u \in \Sigma^*} \mathcal{S}_{Q_2}(v) \\
&= 1.
\end{aligned}$$

- *Induction step*: Let $n \geq 2$. Assume $P(n)$, let us prove $P(n+1)$:

$$\begin{aligned}
\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) &= \sum_{\sigma \in \Sigma^*} \sum_{\sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_n = \sigma} \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i) \\
&= \sum_{\sigma_n \in \Sigma^*} \sum_{\sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_{n-1}} \left(\prod_{i=1}^{n-1} \mathcal{S}_{Q_i}(\sigma_i) \right) \mathcal{S}_{Q_n}(\sigma_n) \\
&= \sum_{\sigma_n \in \Sigma^*} \mathcal{S}_{Q_n}(\sigma_n) \underbrace{\sum_{\sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_{n-1}} \left(\prod_{i=1}^{n-1} \mathcal{S}_{Q_i}(\sigma_i) \right)}_{= 1 \text{ by hypothesis}} \\
&= \sum_{\sigma_n \in \Sigma^*} \mathcal{S}_{Q_n}(\sigma_n) \\
&= 1.
\end{aligned}$$

Choice. Let $Q = \times(Q_1, \ldots, Q_n, p_1, \ldots, p_n)$ with $n \geq 2$:

$$\begin{aligned}
\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) &= \sum_{\sigma \in \Sigma^*} \sum_{i=1}^{n} p_i \mathcal{S}_{Q_i}(\sigma) \\
&= \sum_{i=1}^{n} p_i \underbrace{\sum_{\sigma \in \Sigma^*} \mathcal{S}_{Q_i}(\sigma)}_{=1} \\
&= \sum_{i=1}^{n} p_i \\
&= 1.
\end{aligned}$$

Parallel. Let $Q = \wedge(Q_1, \ldots, Q_n, p_1, \ldots, p_n)$ with $n \geq 2$:

$$\begin{aligned}
\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) &= \sum_{\sigma \in \Sigma^*} \sum_{\sigma_1, \ldots, \sigma_n} \left(\mathbb{P}(Sh(\sigma_1, \ldots, \sigma_n, p_1, \ldots, p_n) = \sigma) \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i) \right) \\
&= \sum_{\sigma_1, \ldots, \sigma_n} \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i) \underbrace{\sum_{\sigma \in \Sigma^*} \mathbb{P}(Sh(\sigma_1, \ldots, \sigma_n, p_1, \ldots, p_n) = \sigma)}_{=1} \\
&= \underbrace{\sum_{\sigma_1, \ldots, \sigma_n} \prod_{i=1}^{n} \mathcal{S}_{Q_i}(\sigma_i)}_{(Sequence) = 1} \\
&= 1.
\end{aligned}$$

Loop. Let $Q = \circlearrowleft(Q_1, Q_2, p)$:

$$\begin{aligned}
\sum_{\sigma \in \Sigma^*} \mathcal{S}_Q(\sigma) &= \sum_{\sigma \in \Sigma^*} \sum_{\substack{m \in \mathbb{N} \\ \sigma_1 . \sigma'_1 . \ldots . \sigma'_{m-1} . \sigma_m = \sigma}} \left(p^{(m-1)} (1-p) \prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \prod_{i=1}^{m-1} \mathcal{S}_{Q_2}(\sigma'_i) \right) \\
&= \sum_{\sigma \in \Sigma^*} \sum_{m \in \mathbb{N}} \sum_{\sigma_1 . \sigma'_1 . \ldots . \sigma'_{m-1} . \sigma_m = \sigma} \left(p^{(m-1)} (1-p) \prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \prod_{i=1}^{m-1} \mathcal{S}_{Q_2}(\sigma'_i) \right) \\
&= \sum_{m \in \mathbb{N}} p^{(m-1)} (1-p) \sum_{\sigma \in \Sigma^*} \sum_{\sigma_1 . \sigma'_1 . \ldots . \sigma'_{m-1} . \sigma_m = \sigma} \left(\prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \prod_{i=1}^{m-1} \mathcal{S}_{Q_2}(\sigma'_i) \right) \\
&= \sum_{m \in \mathbb{N}} p^{(m-1)} (1-p) \sum_{\sigma_1, \ldots, \sigma_m} \left(\prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \right) \underbrace{\left(\sum_{\sigma'_1, \ldots, \sigma'_{m-1}} \prod_{i=1}^{m} \mathcal{S}_{Q_2}(\sigma'_i) \right)}_{(Sequence) = 1} \\
&= \sum_{m \in \mathbb{N}} p^{(m-1)} (1-p) \underbrace{\sum_{\sigma_1, \ldots, \sigma_m} \left(\prod_{i=1}^{m} \mathcal{S}_{Q_1}(\sigma_i) \right)}_{(Sequence) = 1} \\
&= \sum_{m \in \mathbb{N}} p^{(m-1)} (1-p) = 1
\end{aligned}$$

By the induction hypothesis, each sub-sPT Q_i is normalised, that is, $\sum_\sigma \mathcal{S}_{Q_i}(\sigma) = 1$. We have shown that each element of the syntax preserves normalisation. □

6.3 Silent loops and sPTs

In Section 3.4, we pointed out the adverse effect that the presence of loops of silent transitions has on the reachability graph unfolding procedure to compute the language of an sWN model. In the same spirit, we also point out the peculiar effect of silent loops at the sPT level.

Let us consider the sPT shown in (6.2), which represents a loop construct in which the body may execute either the visible action a or the silent action τ, with probabilities p_a and p_{τ_1}, respectively. In the redo branch, the silent action τ is executed with probability p_{τ_2}; otherwise, with probability $1 - p_{\tau_2}$, the loop terminates.

$$Q = \circlearrowleft (\otimes(a, \tau, p_a, p_{\tau_1}), \tau, p_{\tau_2}) \tag{6.2}$$

This process indeed contains a silent loop, since there is a non-zero probability that the action τ is selected both in the body branch and in the redo branch of the loop. In particular, the probability of performing a single silent iteration is $p_{\tau_1} \cdot p_{\tau_2}$. The language $\mathcal{L}_Q$ generated by this process is infinite:

$$\mathcal{L}_Q(\sigma) = \{\varepsilon, \langle a\rangle\, \langle a, a\rangle, \langle a, a, a\rangle, \ldots\} = \bigcup_{k=0}^{\infty} \{\langle a\rangle^k\}$$

Because of the presence of this silent loop, each $\sigma \in \mathcal{L}_Q$ corresponds to an infinite number of possible executions of the process tree. The probability of generating a word of arbitrary length $k \in \mathbb{N}$, that is, the trace $\sigma = \langle a\rangle^k \in \mathcal{L}_Q$ is given, based on the sum of some geometric series, by:

$$P[\langle a\rangle^k] = \frac{p_a^k \cdot p_{\tau_2}^{k-1} \cdot (1 - p_{\tau_2})}{(1 - p_{\tau_1} \cdot p_{\tau_2})^{k+1}}$$

For example:

$$\begin{aligned}
P(\varepsilon) &= p_{\tau_1} \cdot \Big(\sum_{k_1=0}^{\infty} (p_{\tau_2} \cdot p_{\tau_1})^{k_1}\Big) \cdot (1 - p_{\tau_2}) = \frac{p_{\tau_1}(1 - p_{\tau_2})}{1 - p_{\tau_1} p_{\tau 2}} \\
P(\langle a\rangle) &= \Big(\sum_{k_1=0}^{\infty} (p_{\tau_1} \cdot p_{\tau_2})^{k_1}\Big) \cdot p_a \cdot \Big(\sum_{k_2=0}^{\infty} (p_{\tau_2} \cdot p_{\tau_1})^{k_2}\Big) \cdot (1 - p_{\tau_2}) = \frac{p_a(1 - p_{\tau_2})}{(1 - p_{\tau_1} p_{\tau 2})^2} \\
P(\langle a, a\rangle) &= \Big(\sum_{k_1=0}^{\infty} (p_{\tau_1} \cdot p_{\tau_2})^{k_1}\Big) \cdot p_a \cdot \Big(\sum_{k_2=0}^{\infty} (p_{\tau_2} \cdot p_{\tau_1})^{k_2}\Big) \cdot p_a \cdot \Big(\sum_{k_3=0}^{\infty} (p_{\tau_2} \cdot p_{\tau_1})^{k_3}\Big) \cdot (1 - p_{\tau_2}) \\
&= \frac{p_a^2 p_{\tau_2}(1 - p_{\tau_2})}{(1 - p_{\tau_1} p_{\tau 2})^3}
\end{aligned}$$

By numerical computation, it can be verified that $\mathcal{L}_Q$ is a normalised stochastic language:

$$P(\mathcal{L}_Q) = P(\bigcup_{k=0}^{\infty} \{\langle a \rangle^k\} = \sum_{k=0}^{\infty} \frac{p_a^k \cdot p_{\tau_2}^{k-1} \cdot (1 - p_{\tau_2})}{(1 - p_{\tau_1} \cdot p_{\tau_2})^{k+1}} = 1$$

This example shows that the stochastic language generated by an sPT containing silent loops is still normalised. This confirms that the probability assigned to any trace σ correctly accounts for all possible execution paths involving arbitrary alternations of silent and non-silent actions.

6.4 Approximation of an sPT stochastic language

As repeatedly emphasised throughout this thesis, a central aspect of process mining is to assess the conformance between the behaviour observed in the event log and that described by the model. Building directly on the semantics of sPTs, we introduce a stochastic simulation procedure (Algorithm 5) that generates random samples of traces from the stochastic language of a given sPT. This procedure can be employed to obtain arbitrarily accurate approximations of the stochastic language generated by a given sPT, as well as to construct confidence interval estimates for the probability of each word in the language. The recursive function $\textsc{Sample}(Q)$ outlined in Algorithm 5 returns a trace sampled from $\mathcal{S}_Q$.

We denote by $\textsc{RandomChoice}(\{1, \ldots, n\}, (p_1, \ldots, p_n))$ the operation that samples an element from $1, \ldots, n$ according to probabilities $p_1, \ldots, p_n$, and by RandomGeom(p) the operation that samples $m \in 1, 2, \ldots$ from a geometric distribution with parameter p.

Lines 2 - 5: Leaves. In these cases, the procedure terminates immediately, returning with probability 1 either the empty trace ε (if the leaf is labelled τ) or the singleton trace consisting of the activity labelling the leaf. For sPTs whose root is an operator node, the resulting trace is instead constructed recursively. In such cases, whenever the operator involves probability parameters (namely $\times$, $\wedge$, and $\circlearrowleft$), the procedure requires sampling from the corresponding discrete distribution to resolve the choice or interleaving.

Lines 6 - 11: Sequence. In the case of a sequence operator, the algorithm iterates over all child subtrees. For each child Q_i, it recursively samples a trace by invoking the procedure on Q_i, and appends the resulting trace to the current one. In this way, the final trace is obtained as the concatenation of the subtraces, preserving the left-to-right order prescribed by the sequence.

Lines 12 - 15: Choice. For a choice operator, the algorithm randomly selects one of the child subtrees Q_i according to the associated probability distribution $(p_1, \ldots, p_n)$. It then recursively samples a trace from the selected subtree.

Lines 16 - 30: Parallel. For a parallel operator, the algorithm first samples one trace from the language of each child subtree Q_i by recursively calling the procedure.

Algorithm 5 Sampling a trace from an sPT

Require: A stochastic process tree Q
Ensure: A trace $\sigma \in \mathcal{S}_Q$ drawn according to the stochastic semantics of Q

function Sample(Q)
  if $Q = \tau$ **then return** ε
  end if
  if $Q = a$ with $a \in \Sigma_Q$ **then return** $\langle a \rangle$
  end if
  if $Q = \rightarrow (Q_1, \ldots, Q_n)$ **then** // Sequence
    $\sigma \leftarrow \varepsilon$
    for $i \leftarrow 1$ **to** n **do**
      $\sigma \leftarrow \sigma \cdot \textsc{Sample}(Q_i)$
    end for
  end if
  if $Q = \times(Q_1, \ldots, Q_n, p_1, \ldots, p_n)$ **then** // Choice
    $i \leftarrow \textsc{RandomChoice}(\{1, \ldots, n\}, (p_1, \ldots, p_n))$
    return $\textsc{Sample}(Q_i)$
  end if
  if $Q = \wedge(Q_1, \ldots, Q_n, p_1, \ldots, p_n)$ **then** // Parallel
    $\sigma \leftarrow \varepsilon$
    for $i \leftarrow 1$ **to** n **do**
      $\sigma_i \leftarrow \textsc{Sample}(Q_i)$
      $j_i \leftarrow 1$
    end for
    while $\exists\, k : j_k \leq |\sigma_k|$ **do**
      $A \leftarrow \{i \in \{1, \ldots, n\} \mid j_i \leq |\sigma_i|\}$
      $k \leftarrow \textsc{RandomChoice}(A,\ (p_i)_{i \in A})$
      if $j_k \leq |\sigma_k|$ **then**
        $\sigma \leftarrow \sigma \cdot \langle \sigma_k[j_k] \rangle$
        $j_k \leftarrow j_k + 1$
      end if
    end while
  end if
  if $Q = \circlearrowleft (Q_1, Q_2, p)$ **then** // Loop
    $\sigma \leftarrow \textsc{Sample}(Q_1)$
    $m \leftarrow \textsc{RandomGeom}(p)$
    for $i \leftarrow 1$ **to** $m - 1$ **do**
      $\sigma \leftarrow \sigma \cdot \textsc{Sample}(Q_2) \cdot \textsc{Sample}(Q_1)$
    end for
  end if
  return σ
end function

The obtained subtraces must then be interleaved in order to form the final trace. This is achieved through a while loop that maintains, for each subtrace, an index pointing to the next unread activity. At each iteration, the algorithm considers the set A of active subtraces, i.e., those for which unread activity remain. The next contributing subtrace $k \in A$ is then selected by sampling according to the original weights $(p_1, \ldots, p_n)$ renormalized over the active set A, namely

$$\tilde{p}_i = \frac{p_i}{\sum_{j \in A} p_j} \qquad \text{for each } i \in A.$$

The next activity of σ_k is appended to the growing trace, and the index j_k is advanced. This procedure is repeated until all subtraces have been entirely consumed.

Lines 31 - 37: Loop. For a loop operator, the algorithm first samples a trace from the body (left child) Q_1, which initiates the construction of the final trace. Then, it samples a value $m \in \{1, 2, \ldots\}$ from a geometric distribution with parameter p, where p represents the probability of repeating the loop after each execution of the body. Equivalently, the loop terminates with probability $(1-p)$ after each iteration. The distribution of m is therefore given by

$$\mathbb{P}[m = k] = p^{k-1}(1-p) \qquad \text{for } k \geq 1$$

The sampled value m determines the total number of body executions: the loop iterates $m-1$ times over the loop condition (right child) Q_2, followed by Q_1, and finally performs one last execution of the body Q_1 before termination.

6.5 Stochastic process tree optimisation

In previous chapters we have argued the relevance of discovering optimal (weight) parameters for an sWN model and presented two complementary approaches to tackle this problem: an optimisation framework for discovery of optimal weight parameters (Chapter 4) that exploit exact computation of the sWN language via RG unfolding (Chapter 3) and a Bayesian parameter inference method which is based on simulation-based approximations of the sWN language (Chapter 5).

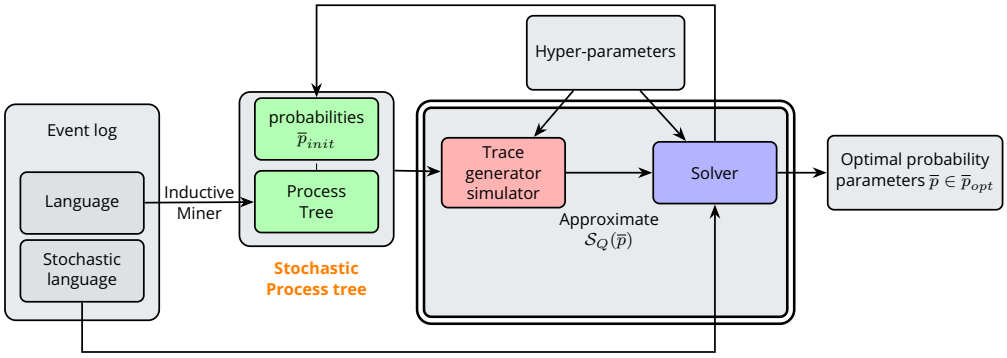


Figure 6.11 – Discovery of optimal sPT parameters

Having introduced the sPT formalism, we now address the problem of discovering optimal parameters for an sPT model obtained from a given log. We do so

Event log	log properties		sWN	sPT	sPT-based optimisation via simulation				sWN-based optimisation (see Chapter 4)	Burke's estimation [21]	WaWE [17]
	#traces	#act	#pars	#pars	N	Initial sPT rEMD	Optimized sPT rEMD	runtime (s)	rEMD	rEMD	rEMD
BPIC13_c	183	4	19	10	1,000 10,000 25,000 50,000 100,000	0.2444 0.2361 0.2283 0.235 0.2327	0.113 0.0892 0.0891 0.088 0.0854	144 496 1223 1988 3103	0.04	0.63	0.21
BPIC13_i	1511	4	23	11	1,000 10,000 25,000 50,000 100,000	0.62404 0.62330 0.64473 0.64368 0.64173	0.44216 0.28088 0.23670 0.23625 0.23647	3525 3641 4322 5487 13163	0.18	0.70	0.69
BPIC13_o	108	3	20	9	1,000 10,000 25,000 50,000 100,000	0.2666 0.2562 0.2606 0.2583 0.2585	0.2642 0.1042 0.0926 0.0891 0.082	11 598 1551 3224 6692	0.076	0.24	0.26
BPIC17_ol	16	8	11	7	1,000 10,000 25,000 50,000 100,000	0.1953 0.1973 0.2057 0.2002 0.1918	0.0729 0.0326 0.0271 0.0193 0.0458	7 70 265 968 2263	0.09	0.09	0.08
Roadfines	231	11	34	21	1,000 10,000 25,000 50,000 100,000	0.3594 0.3482 0.349 0.3485 0.3467	0.2124 0.1203 0.0625 0.0986 0.0377	171 809 2626 3268 8020	0.08	0.27	0.48

Table 6.1 – sPT-based process discovery on real-life event logs compared with other approaches

based on the simulation procedure given in Algorithm 5, which we use to obtain an approximation of an sPT language issued by the considered sPT and given a parameter vector.

The optimisation procedure (outlined in Figure 6.11) begins with an input event log from which its language (the set of observed traces) is extracted. A process tree capable of reproducing all traces in the log is then discovered using the Inductive Miner algorithm. This tree captures the control-flow structure of the process. Next, initial parameters are randomly sampled and assigned as probabilities to the probabilistic operators present in the tree (i.e., choice, parallel, and loop nodes), resulting in an initial sPT. These parameters are then numerically optimised iteratively. The goal of the optimisation is to minimise the distance between the SL generated by the sPT and the SL derived from the event log. At each iteration, the optimisation engine approximates the stochasticity of emissions by the sPT (with given probability vector $\overline{p}$) by sampling n executions of the sPT via Algorithm 5. The process continues until the rEMD distance between the log's and the sPT languages converges below a chosen upper bound ε.

6.5.1 Experiments

To evaluate the sPT optimisation framework, we conducted a series of experiments using real-world event logs of varying complexity. Table 6.1 summarises the relevant statistics concerning the considered logs: column *#traces* reports the number of distinct traces, and column *#act* the number of distinct activities. Column *sPT_#pars* shows the number of parameters in the mined sPT. To highlight the parameter reduction achieved by the sPT formalism, column *sWN_#pars* reports the number of parameters (weights) in the corresponding sWN models. In practice,

sPTs reduce the parameter count by approximately 40%-50% compared to sWNs.

Optimisation of the sPT parameters was carried out by varying the number of sampled traces N (column *N*) used to approximate the stochastic language at each iteration. As initial parameters, we retained the configuration with the best rEMD among 10 randomly generated ones. In light of the non-differentiability of rEMD (see Section 4.2.2), we employed Powell's method, which is particularly well suited for objectives where gradient-based techniques are less effective. Columns *sPT-based optimisation via simulation* report the initial rEMD, the final rEMD after optimisation, and the corresponding execution times. As expected, increasing N consistently improves the final rEMD values, though at the cost of longer runtimes.

Table 6.1 also compares our approach against three alternative methods, all of which operate on sWNs. Although these methods involve a larger number of parameters, thus making optimisation more complex, they offer greater modelling flexibility. For each method, we report the rEMD of the optimised sWN. Since they are not simulation-based, the parameter N does not apply. The column *sWN-based optimisation* corresponds to the approaches described in Chapter 4, which optimise weights by minimising rEMD using an unfolding technique to compute the stochastic language of the sWN exactly. Despite the significantly smaller number of parameters, sPTs outperform these approaches on two of the logs. The column *Burke's estimation* refers to the work of [21], which defines six weight estimators by combining simple log statistics with structural features of the mined sWN. We report in the table the estimator yielding the best rEMD value, excluding the alignment-based estimator. While Burke's method is computationally very efficient, it provides a poor approximation of the log's stochastic language. Finally, the column *WaWE* compares our approach to the method in [17], which optimises weights using subgradients of the Earth Mover's Stochastic Conformance measure. Since WaWE requires tuning five hyperparameters that strongly affect its performance, we relied on the authors' reference implementation. The rEMD values obtained are substantially worse for most logs, which can largely be explained by the fact that the optimised measure is not precisely equivalent to rEMD.

All experiments were conducted using the ProDiSt prototype tool, publicly available together with the event logs at https://github.com/DocPierro/ProDiSt. For optimisation, the tool relies on the `scipy.optimize` package, which provides several optimisation algorithms.

6.6 Conclusion

In this chapter, we introduce the stochastic extension of process trees, a novel formalism that provides a more compact alternative to stochastic workflow nets. Starting from the definition of the syntax, which is obtained by enriching process tree operators (sequence, choice, parallel, loop, and leaf) with probabilistic parameters, we then formalised the semantics by outlining the stochastic language issued by the stochastic extension of each tree operator. Based on the sPT semantics,

we then defined a simulation-based procedure for sampling traces of a given sPT model, demonstrating that such a simulation procedure can be used to obtain an arbitrarily precise approximation of the stochastic language generated by an sPT model. Within an optimised discovery framework that relies on methods to approximate its stochastic language.

Compared to stochastic workflow nets, sPTs offer a more structured representation where concurrency and repetition patterns are made explicit. This facilitates both the analysis of stochastic behaviours and the design of algorithms that exploit the tree structure. However, challenges remain regarding the tractable treatment of infinite behaviours induced by loops and the combinatorial explosion of parallel interleavings.

Altogether, stochastic process trees provide a unifying and flexible formalism for reasoning about probabilistic process models. They lay the foundation for discovery techniques that operate directly on tree-structured models, as well as for transformations into other stochastic formalisms, such as workflow nets or stochastic Petri nets.

Chapter 7

Conclusion

The ability to account for the probabilistic nature of a real-life process is of paramount importance within the process mining domain, especially when quantitative analysis of performance metrics is a relevant factor in validating a process. Suppose classical process discovery has proven a successful story leading to many effective methods [71, 86, 49, 76] for extracting models capable of capturing control-flow aspects contained in the observed executions of a process. In that case, it inherently falls short when considering the stochastic dimension of the phenomenon.

Such a deficiency has led to a growing interest in the research community in the stochastic extension of the process discovery problem, with several contributions being presented in recent times [21, 22, 17, 45]. Accounting for the probabilistic dimension of a log means deriving a stochastic model that reliably reproduces not only the control-flow aspects of the observed traces but also the likelihood with which they have been observed. Independent of the considered formalism, stochastic process models are characterised by a vector of parameters that affect the stochastic character they exhibit. More concretely, the parameters of a stochastic model govern how the probability mass is distributed over the words of the language they emit.

In this respect, then, assuming a control-flow model is given, the core of the stochastic process discovery problem consists of identifying optimal parameters such that the corresponding stochastic extension of the model distributes the probability mass over the words it generates in a way similar to how the probability mass is distributed over the traces in the log. In other words, stochastic process discovery boils down to an optimisation problem based on an objective function that accounts for similarity between (discrete) probability distributions, such as, for example, the Earth Mover's Distance [52, 53] or the entropy relevance [63, 46, 5].

The discovery of optimal parameters for a stochastic process model is hindered by two correlated pitfalls that affect the complexity of the task: determining the

model's stochastic language and the dimension of the parameter space (i.e., the number of parameters in the model). Clearly, the more complex the model is, the larger its parameter space and the harder it is to assess its language.

In an effort to contribute to advancements in tackling the stochastic process discovery problem, this thesis presents a number of contributions addressing the pitfalls that undermine this problem. Specifically:

Exact computation of an sWN language. In the realm of stochastic workflow nets, arguably the most popular class of models used in process mining, we introduced a method for computing the exact probability of the traces emitted by an sWN model instance (i.e., an sWN equipped with a specific vector of weight values for its transitions). This approach is based on a breadth-first traversal of the sWN reachability graph; hence, it allows for exhaustively unfolding all traces emitted by an sWN while propagating the numerical computation of the probability of the unfolded traces, on the fly, as they are unfolded. To ensure termination, the proposed method can be adapted to take into account different stopping criteria, such as halting the unfolding as soon as the mass of probability of the (currently) unfolded traces meets a given lower bound. Since, in our implementation, we assumed the procedure would operate on perfectly fitting models, we opted to terminate the unfolding as soon as all traces of the log have been unfolded. Therefore, in practice, given a log, a perfectly fitting WN model obtained from it and a specific vector of weights for the model's transitions we established a computational framework through which one can compute the exact probability with which the log's traces occur in the model. Notice that in most cases the probability that the log's traces have to occur in the model do not sum to 1, therefore a normalisation step is necessary in order to plug the result of such computation within an optimisation framework. The main advantage of this method is that, unlike approaches aimed at deriving symbolic expressions for the probability of an sWN transition [45], which suffer from a major scalability issue, it has proven effective on models extracted from commonly used benchmark logs. To improve efficiency, we then developed an evolved unfolding procedure through which, rather than storing the numerical values of the probability of each trace being unfolded, we store the function (of the weight parameters) that allows us to compute the corresponding probability value given a vector of weight values. This allows for avoiding the reiteration of unfolding when evaluating the trace probabilities for different weight vectors. When the DAG is built once for a given sWN, it can be reused to retrieve the language induced by a specific weight vector (also allowing to avoid re-iterating identical computations thanks to memoisation). Despite such much improved computational efficiency, the main issue of the unfolding approach, on the other hand, is in terms of space complexity, as the size of the DAG constructed via unfolding can explode depending on the characteristics of the log (hence of the structure of the sWN), limiting the applicability of the method on complex sWN models.

Discovery of optimal weights for an sWN model. Relying on the exact computation of an sWN language, we then defined a framework to discover optimal weights

for a given sWN model. To this aim, we considered two types of objective functions. The first one is a customised version of the Earth Mover's Distance (EMD), which we named restricted EMD (rEMD) as it only accounts for the intersection between the model's and the log's traces while computing the minimal amount of probability mass to be moved. The second is a differentiable function, the log-likelihood function, derived from maximum likelihood estimation, which evaluates the probability that the sWN generates the observed event log. The optimisation scheme will rely on both gradient-based and non-gradient-based solvers to explore the parameter space and refine the weight vector assigned to the sWN, aiming to minimise one of the metrics. At each step, the solver proposes a new weight vector, and the model's conformance is evaluated by applying the unfolding procedure.

Simulation-based inference of optimal weights for an sWN model. To overcome the limitations of potentially costly numerical computations in an sWN language, we have considered applying a simulation-based alternative. Classically, the advantage of employing a simulation-based approach is that one can trade precision for runtime. Therefore, we introduced a novel method that, relying on statistical model checking, allows us to obtain an arbitrarily precise approximation of the probability of the traces generated by an sWN. Such an approximation is then combined with a Bayesian parameter inference scheme that enables an efficient exploration of the parameter space, driven by a given stochastic conformance criterion (in our implementation, the rEMD). We have validated the approach through experiments run using workbench event logs, showing that for several logs it outperforms the exact computation-based optimisation method, that is, identifying transition weights, which results in smaller rEMD (with comparable runtime).

Reducing the parameter space dimension by means of the stochastic process tree formalism. A critical aspect of WN models discovered from a log is that, as a consequence of the discovery strategy, they often turn out to be filled with many redundant silent transitions. Such transitions can often be considered as redundant because, other than sensibly playing a role in allowing to reproduce the control-flow characteristics of the log's traces, they also induce a potentially large number of interleavings for given traces that the sWN model emit (i.e. due to redundant silent transitions there may be several ways for an sWN model to generate a very same trace). This is certainly the case for nets generated via the inductive miner algorithm, which indeed result from a mapping step through which a discovered process tree is translated into an equivalent WN. To mitigate this downside, we introduced the stochastic extension of process trees, which is the source formalism from which WN is derived within the inductive miner. The advantage of stochastic process trees is twofold. On one hand, they reduce the number of parameters that the models depend on (which is clearly beneficial when it comes to searching for optimal parameters) and, perhaps even more importantly, they eliminate the ambiguity of redundant silent sWN transitions. Based on the formalisation of the sPT syntax and semantics, we then defined a stochastic simulation procedure that allows for sampling traces emitted by an sPT model. Based on this, we developed a

simulation-based engine to discover optimal parameters for an sPT model.

7.1 Scientific positioning

This section positions the proposed work with respect to three closely related families of approaches: Burke's stochastic estimators, WaWE, and the SLPN Miner.

Although all methods presented in this thesis rely on optimisation to infer probabilistic behaviour, the rationale behind Burke's estimators differs substantially. Burke's approach relies on the relationship between trace patterns observed in the log and how these patterns are represented in the workflow net, without performing global optimisation on model parameters. Instead, it computes stochastic estimators derived from the structural correspondence between traces and model constructs, whereas our methods explicitly optimise a distance between the model's induced stochastic language and the log.

On the other hand, WaWE and the SLPN Miner are conceptually closer to the unfolding-driven optimisation methods introduced in this thesis, as they also rely on explicit optimisation schemes to infer probabilistic information. WaWE formulates the learning of transition weights as an optimisation problem and employs subgradient ascent to maximise the model conformance measure. Similarly, the SLPN Miner uses classical global optimisation techniques in practice to estimate transition weights within the discovered stochastic workflow net. Below we summarise the principle differences between the WaWE approach, the SLPN Miner approach and the various approaches for stochastic process discovery introduced in this manuscript. We do so w.r.t. two perspectives: the conformance criteria used to drive the optimisation process and the method used to determine the stochastic language generated by a model instance.

- **Conformance metric.** Each of the above mentioned methods employ a different conformance metric as its objective function, depending on its specific needs and methodological requirements. When it comes to WaWE, which builds upon the EMSC score, a well-known drawback of EMSC is that it allows optimisation procedures to exploit probability reallocation. When the model admits many (possibly infinite) alternative traces, the objective can be improved by shifting probability mass away from log traces toward nearby model traces not present in the log, without actually improving the similarity between the stochastic languages. WaWE addresses this issue by relying on a penalised EMSC formulation, which explicitly penalises probability assigned to model traces outside the log, thereby preventing such degenerate reallocations and stabilising the optimisation. The SLPN Miner addresses this issue by relying either on the unit EMSC or on the entropy relevance measure. In contrast, all methodologies introduced in this manuscript avoid this problem by operating exclusively on the restricted language of the log through the use of the so-called restricted Earth Mover's Distance or the log-likelihood. These

strategies also come with drawbacks, the first being that requiring perfect fitness for the model is a strict and often difficult condition to satisfy. The second drawback is that they introduce a bias in the metric: by completely ignoring model behaviours that are not supported by the log, and by not applying any penalisation to those behaviours, the resulting distance is necessarily biased in a way that is difficult to quantify.

- **Determination of the stochastic language of an sWN instance.** A more pronounced difference lies in a crucial aspect of the overall strategy: the method used to extract the stochastic language of the sWN with respect to a given weight assignment. Since the optimisation step requires computing these stochastic languages a large number of times, the extraction procedure must be as efficient as possible. WaWE operates through the simulation of a fixed number of net paths. Simulation is one of the fastest and simplest ways to approximate the stochastic language, but the quality of this approximation has a significant impact on the effectiveness of the optimiser. Poorly approximated languages can misguide the optimisation process, leading either to slow convergence or to sub-optimal solutions. The SLPN Miner uses a series of automaton structures, one for each trace in the log, that symbolise the behaviour of each log trace. Then, through a cross-product with the reachability graph of the net, it obtains a compact and highly effective representation of the underlying equation system induced by the stochastic semantics. However, the main limitation of this representation is that the cross-product can grow rapidly as the size and complexity of the log language increase. As a result, the approach quickly becomes infeasible when dealing with real-life event logs that require perfect fitness and contain a vast number of distinct traces. The unfolding strategy can be compared to this cross-product construction, as it also explores the reachability graph in detail, guided by the traces of the log, and the resulting DAG contains all the information required to reconstruct the corresponding system of equations. The difference, however, lies in the computational profile. This DAG is significantly more expensive to construct for logs of reasonable size than the automaton-based structure used by the SLPN Miner; however, it scales much better as the log grows larger or contains highly diverse behaviour.

When it comes to ABC-SMC model inference, the contribution is of a very different nature. ABC-SMC can efficiently explore the weight-vector space and identify regions that lead to models whose stochastic language closely matches that of the log, without requiring explicit gradients or analytical optimisation objectives. The simulation strategy used in this context, based on statistical model checking, is also novel but remains conceptually close to common sWN simulation approaches used in the community, such as the path-simulation procedure employed by WaWE.

Finally, approaching the discovery problem through the optimisation of stochastic process trees can be compared to the strategy employed in the Toothpaste Miner. The definition of probabilistic process trees in that work is very close to the one adopted in this thesis. The main difference lies in how stochasticity is expressed:

the Toothpaste Miner assigns weights to nodes and leaves to emulate the spirit of sWNs, which makes the resulting structure easy to transform into an sWN, whereas the sPT formalism used in this thesis works directly with probabilities, which simplifies the semantics and makes the computation of the stochastic language more tractable. This difference arises because the Toothpaste Miner treats the process tree as an intermediate step in discovering an sWN. In contrast, in this thesis, the sPT is considered an end in itself for the time being (the problem of defining translation rules to an equivalent sWN model has not been treated in this thesis). Beyond this, the sPT is discovered through an optimisation step that aims to maximise its conformance to the log, whereas the Toothpaste Miner relies on predefined reduction rules to refine the complexity of the tree while maintaining a conformant model.

7.2 Future perspectives

Despite the advances represented by the contributions discussed above, several extensions and open issues remain for future work in stochastic process discovery.

Extending the experimental evaluation. A relatively straightforward direction for future work is to broaden the experimental evaluation of all proposed methods to a larger and more diverse set of event logs, varying in size and complexity. We also aim to assess the behaviour of our approaches on specific real-life case studies, as well as on synthetic logs intentionally designed to exhibit atypical or challenging behaviours, such as deeply nested loops, intricate concurrency patterns, or rare exceptional traces. Such experiments would help identify shortcomings that cannot be observed within the current benchmark set.

Exploring alternative stochastic conformance measures. In the same spirit, we plan to extend our methods to support additional stochastic conformance and distance metrics, such as the entropic relevance [46], which has proven to be one of the most effective and widely adopted approaches for quantifying similarity between stochastic languages.

Handling silent transitions in the unfolding of sWNs. Throughout this work, one of the most significant challenges encountered when dealing with sWNs concerns the presence of silent τ labelled transitions. The drawbacks introduced by such transitions were, in fact, the initial motivation that led us to explore the sPT formalism. In the context of the unfolding procedure described in Chapter 3, silent transitions are the leading cause of the exponential growth in the size of the generated and explored DAG. In the short term, we are investigating improvements to the unfolding procedure aimed at effectively "removing the silence" from the workflow net's reachability graph. The idea is to derive from the reachability graph annotated with switching-marking probabilities a reduced unfolding DAG that omits redundant silent behaviour. In a subsequent step, we plan to represent the reachability graph as a stochastic transition matrix capturing the likelihood of moving between mark-

ings, and to perform a matrix decomposition to separate the contribution of each transition. This would allow us to measure the influence of silent transitions on the overall stochastic behaviour of the net, and ultimately handle them more effectively.

Better exploitation of the ABC–SMC marginal posterior distribution of optimal weight parameters. Most indirect stochastic discovery methods are designed to return a single local optimum in the stochastic parameter space, providing limited insight into the broader landscape of possible solutions. In contrast, the ABC–SMC approach yields a more informative outcome, producing an approximation of the full posterior distribution over the weight parameters. Henceforth, we actually use only a small amount of this information for the discovery. The posterior not only identifies regions of high probability, which correspond to near-optimal parameter configurations, but also reveals correlations between weights and potential multimodality of the solution space. From now on, this eventual multimodality, if it exists, cannot be observed, as the only law studied from weight is a customary normal truncated law [16]. Future work will focus on better exploiting additional information, for instance, to learn more about the impact of each transition in the net, to detect alternative stochastic behaviours consistent with the observed log, or to initialise and guide other optimisation procedures.

Exact computation of a stochastic process tree language. The indirect discovery of sPTs currently relies on a simulation-based optimisation procedure. We argue that such an optimisation process would benefit from an exact computation of the stochastic language rather than a sampled approximation, as this would provide a more precise guidance to the solver in identifying optimal parameter values. We are therefore working on a traversal procedure for sPTs that enables the exact computation of their stochastic language. However, the high combinatorial complexity induced by concurrency and loops, resulting in a potentially vast number of possible traces, makes this task particularly challenging. For this reason, the procedure must initially remain log-driven, similarly to the unfolding approach described in Chapter 3. The envisioned method follows a bottom-up strategy in which each subtree recursively generates only the subset of traces necessary for the root to reconstruct the stochastic language restricted to those observed in the log. Significant difficulties arise with intricate structures that combine multiple loops and concurrency, as identifying the relevant traces without resorting to brute-force enumeration remains highly challenging, yet necessary to keep computation times reasonable.

Translation of sPTs into stochastic workflow nets. The current outcome of the stochastic discovery method described in Chapter 6 is an optimised sPT. However, at present, we lack a systematic way to translate the semantics of sPT operators into a more widely implemented formalism, such as sWNs. First, we are actively investigating a stochastic extension of the mapping between process trees and workflow nets introduced initially in the Inductive Miner paper [49], to enable the automatic generation of an equivalent sWN from a given sPT. Preliminary attempts

have shown that this translation is far from trivial and may even be impossible in general, as preserving the exact stochastic semantics would require finding a consistent mapping between the probability parameters of the tree nodes and the weight parameters of the transitions. Achieving such equivalence might further necessitate structural adaptations of the resulting workflow net to reproduce the semantics of the original sPT faithfully. Second, we are working on the formal translation of the sPT formalism into languages commonly used in the probabilistic model checking domain, such as the PEPA process algebra [37] and the Reactive Modules language [7]. Both are supported by the PRISM model checker [43] and allow for high-level specifications of Markov chains. This translation would enable the expressive and rigorous formal analysis of sPT models using (stochastic) temporal logic properties.

In conclusion, stochasticity in process behaviour is essential to extend the reach of process mining beyond its current boundaries. It extends the expressive power of models, enabling a more faithful and nuanced representation of real-world dynamics. At the same time, it naturally leads to the exploration of another fundamental and even more intricate aspect of reality: the temporal dimension. So far, timestamps in event logs have mainly been used within structural discovery approaches to determine the execution order of activities within traces. However, this temporal information can be exploited far beyond ordering and can be used to model and analyse the waiting times between activities, providing insights into process performance and resource behaviour. Incorporating time as an explicit quantitative dimension would open the door to predictive analyses, such as estimating remaining processing times, identifying bottlenecks, and forecasting process outcomes. Bringing stochastic and temporal perspectives together thus represents a promising next step toward more realistic and expressive process mining models. As a future direction, building on early works [73], we plan to explore the problem of explicitly modelling time within process models further. This involves extending existing formalisms, such as sWNs and sPTs, with temporal parameters, and leveraging our optimisation-based discovery methods to estimate those time-related parameters so that they best fit the event log, ultimately capturing more of the fundamental dynamics embedded in the data.

Chapter 8

French Summary

Avec le temps, les données sont devenues une composante de plus en plus centrale de notre quotidien, contribuant à façonner la manière dont nous comprenons, analysons et améliorons des systèmes complexes. Chaque événement, qu'il soit déclenché par une interaction délibérée ou qu'il survienne spontanément, constitue un fragment d'histoire qui, une fois assemblé, permet de révéler la dynamique sous-jacente des systèmes observés. La reconstruction et l'analyse de ces histoires se situent au cœur de la science moderne fondée sur les données. Le *process mining* [74] est un domaine de recherche qui vise à extraire des connaissances pertinentes à partir de ces fragments. Il propose des méthodes permettant de collecter et de structurer l'information, de reconnaître des motifs de comportement et, finalement, de construire des modèles concis et interprétables capables de capturer les caractéristiques essentielles d'un processus dans son ensemble. À partir de ces représentations, il devient possible d'exploiter les expériences passées afin de prédire et d'améliorer le fonctionnement futur de ces systèmes.

Au-delà de la description structurelle des processus, un défi plus subtil et de plus en plus important du *process mining* réside dans l'exploitation de l'information stochastique naturellement contenue dans les données [48]. Cette information probabiliste reflète la vraisemblance d'observer certains comportements au cours de l'exécution du système. Lorsqu'elle est correctement modélisée, elle ouvre une nouvelle dimension d'analyse et peut influencer de manière significative les enseignements tirés par les analystes de processus et les experts métier. Bien que l'utilisation de modèles stochastiques en *process mining* ne soit pas nouvelle, le développement d'approches efficaces de découverte de processus stochastiques, capables de produire des modèles reproduisant fidèlement le caractère stochastique du journal considéré, demeure à ce jour une branche de recherche relativement jeune. Elle suscite toutefois un intérêt croissant au sein de la communauté et est aujourd'hui identifiée comme un défi majeur du domaine [3]. La difficulté de la découverte d'un processus stochastique réside dans le fait que les modèles résultants doivent

non seulement être capables de reproduire les exécutions observées, c'est-à-dire l'ordre dans lequel les événements se sont produits, mais également les probabilités avec lesquelles elles ont été observées. Étant donné que plusieurs paramètres probabilistes caractérisent les modèles de processus stochastiques et influencent la vraisemblance des exécutions qu'ils génèrent, la découverte d'un processus stochastique implique, entre autres, la détermination de valeurs optimales pour ces paramètres au regard d'un critère de conformité stochastique donné.

Cette thèse s'intéresse ainsi au problème de l'extraction d'un modèle stochastique capable de reproduire de manière optimale le caractère stochastique des systèmes étudiés.

8.1 Processus et Process Mining

Un processus métier [44] est constitué d'un ensemble d'activités visant à atteindre un objectif organisationnel spécifique. La compréhension des processus métier est cruciale pour les organisations à plusieurs titres : d'une part, pour obtenir une description formelle de processus souvent peu ou pas documentés, et d'autre part, pour exploiter les modèles de processus obtenus afin de surveiller, analyser et éventuellement améliorer le processus étudié.

Au cœur du *process mining* se trouve le journal d'événements, qui fournit les données observationnelles brutes décrivant la manière dont les processus se déroulent et interagissent au sein de systèmes réels. La transformation de ces journaux en représentations pertinentes, capables de soutenir la compréhension, le diagnostic et l'amélioration des processus, nécessite des méthodologies rigoureuses et bien définies. Le *process mining* répond à ce défi en exploitant ces collections d'événements afin de mettre en évidence la structure et le comportement sous-jacents des processus métier.

Les techniques de *process mining* sont généralement classées en trois grandes catégories [2] :

- *La découverte de processus (process discovery)*, qui consiste à générer automatiquement un modèle de processus à partir d'un journal d'événements, sans s'appuyer sur un modèle préexistant ;
- *La vérification de conformité (conformance checking)*, qui compare un journal d'événements à un modèle de processus existant afin d'évaluer dans quelle mesure le comportement observé est conforme au modèle ou s'en écarte. Elle fournit des diagnostics permettant d'identifier les comportements ou activités non conformes et peut quantifier le degré de similarité entre le journal et le modèle à l'aide de plusieurs métriques, telles que l'adéquation (*fitness*) ou la précision (*precision*) [24, 4] ;
- *L'enrichissement (enhancement)*, qui vise à affiner ou étendre un modèle de processus existant en y intégrant des perspectives supplémentaires extraites du journal. Cela inclut notamment l'enrichissement du modèle par des indica-

teurs de performance (par exemple, les goulets d'étranglement ou les temps d'attente), des informations relatives aux ressources, ainsi que l'amélioration de ses capacités descriptives et prédictives [75, 74].

8.1.1 Les journaux d'événements : une unité fondamentale pour la capture de l'information dans les processus métier

En pratique, un journal d'événements [74] est représenté comme une collection d'événements, où chaque événement capture une interaction entre le système et son environnement. Il spécifie généralement l'activité réalisée, son horodatage, les ressources impliquées, ainsi que des informations contextuelles supplémentaires. L'environnement peut inclure aussi bien des acteurs humains (par exemple, des employés traitant des dossiers ou des patients suivant des procédures) que des composants automatisés tels que des capteurs, des machines ou des services logiciels.

Ce qui est particulièrement pertinent pour la découverte, l'évaluation et l'enrichissement des processus décrits par un journal d'événements est de considérer les exécutions observées du processus comme un ensemble de traces. Chaque trace correspond à une séquence d'événements horodatés décrivant l'évolution d'un cas unique. Étant donné qu'une même séquence d'activités peut se produire dans plusieurs cas, un journal d'événements peut contenir plusieurs occurrences de traces identiques. La fréquence associée à chaque trace quantifie le nombre de fois où une séquence d'actions donnée est observée et reflète ainsi la vraisemblance du comportement correspondant au sein du processus. L'ensemble des traces, conjointement à leurs fréquences, constitue un langage décrivant les comportements enregistrés dans le journal d'événements. Dans le même esprit, en normalisant les fréquences des traces, on obtient une distribution de vraisemblance empirique sur l'ensemble des traces, donnant naissance à un langage stochastique. En parcourant l'intégralité du journal, il est alors possible d'extraire chaque séquence d'activités distincte et d'associer à chacune le nombre de fois où un cas est apparu.

8.1.2 Modèle de processus : une représentation compacte et exploitable des processus métier

Selon l'approche et l'objectif retenus, le modèle de processus issu d'un journal d'événements peut prendre différentes formes, allant d'objets mathématiques, tels que des systèmes de transition comme les automates ou les réseaux de Petri [41], à des représentations graphiques plus globales et orientées expertise, comme les diagrammes UML [68]. Ils peuvent également se présenter sous la forme de contraintes déclaratives définies sur les relations entre activités [56]. Quel que soit le formalisme considéré, tous les modèles de processus partagent un objectif fondamental commun : fournir une représentation compacte, structurée et interprétable du comportement observé dans le processus. Ils décrivent la manière dont les activités

du journal sont liées entre elles, capturant ainsi le flot de contrôle et, dans certains cas, les perspectives liées aux données et aux ressources du processus.

Les réseaux de Petri constituent en particulier un formalisme de modélisation des processus particulièrement puissant, offrant une expressivité notable pour représenter les conflits et la concurrence [1]. Un réseau de Petri est composé d'un ensemble de transitions, étiquetées par des activités issues du journal d'événements, dont le tirage représente l'occurrence de l'activité correspondante dans le système. Les transitions sont entourées de places, qui contiennent des jetons et sont reliées aux transitions par des arcs. L'ensemble formé par les places, les transitions et les arcs définit les relations causales et conditionnelles qui gouvernent quand et comment les transitions peuvent se déclencher, permettant ainsi de capturer l'ensemble des comportements dynamiques que le processus peut produire. La définition formelle et la sémantique des réseaux de Petri sont détaillées au Chapitre 2. La structure des réseaux de Petri est naturellement définie de manière à pouvoir être étendue à des dimensions stochastiques. Lorsque des transitions sont en conflit, l'affectation de paramètres de poids permet de spécifier leur vraisemblance. En combinant les probabilités des transitions impliquées dans une exécution donnée, il est alors possible de dériver la vraisemblance du comportement correspondant.

8.2 Objectifs de recherche et problématique

Le problème central abordé dans cette thèse est la découverte automatique de modèles de processus qui soient non seulement structurellement corrects, mais également stochastiquement fidèles. Cela implique la définition de formalismes de modélisation appropriés, le développement d'algorithmes efficaces permettant d'extraire à la fois la structure et les probabilités d'un modèle stochastique, la conception de techniques d'évaluation de la conformité, et enfin la mise au point de procédures permettant la découverte effective de tels modèles en pratique.

8.2.1 Motivation

Comme discuté précédemment, l'aspect stochastique d'un processus peut révéler des informations cruciales sur le fonctionnement du système étudié. L'interprétation de ces informations dépend du domaine d'application, mais, de manière générale, les données stochastiques mettent en évidence les comportements les plus probables ainsi que ceux qui sont plus rares ou exceptionnels.

À un premier niveau d'analyse, cela permet d'évaluer si les traces les plus probables correspondent aux comportements attendus et souhaitables du système et de son environnement, et si les traces les moins probables reflètent des exceptions rares mais valides, ou au contraire du bruit ou des anomalies présentes dans le journal [15, 40]. À un niveau d'analyse plus approfondi, ces informations peuvent également aider à isoler certaines composantes du système, telles que les ressources humaines ou matérielles qui sont surutilisées, sous-utilisées ou inutilisées [35].

8.2.2 Questions de recherche

Cette thèse s'attaque à plusieurs défis majeurs liés à la découverte de modèles de processus stochastiques capables de reproduire de manière précise et efficace les comportements observés, dans le cadre d'approches indirectes d'estimation de paramètres et de stratégies d'inférence bayésienne. Pour répondre à ces questions, nous proposons des méthodes combinant analyse de modèles, techniques d'optimisation et inférence statistique, afin de mettre au jour la nature probabiliste des processus à partir de données événementielles.

(Q1) Étant donné un modèle de processus stochastique, est-il possible de calculer, exactement ou approximativement, l'ensemble des traces qu'il peut générer ainsi que leurs probabilités associées ?

De nombreuses applications requièrent une extraction rapide et précise du comportement d'un modèle de processus découvert. L'usage le plus immédiat consiste à évaluer la conformité du modèle par rapport au journal d'événements original. Une seconde motivation concerne l'amélioration du modèle, où l'objectif est d'accroître son alignement avec le journal observé. Au-delà de la conformité et de l'amélioration, l'extraction du comportement probabiliste d'un modèle est également un prérequis pour des tâches telles que la prédiction de performances, l'analyse des ressources ou l'évaluation des risques. Dans l'ensemble de ces contextes, il est nécessaire de disposer de procédures efficaces permettant de calculer le comportement probabiliste du modèle, c'est-à-dire les probabilités qu'il assigne à des ensembles de traces. Selon le cadre applicatif, ces calculs peuvent être exacts, lorsque l'on exige une précision complète, ou approximatifs, lorsque l'on doit évaluer rapidement un grand nombre de modèles candidats. Nous explorons ces deux possibilités :

- *Méthodes exactes.* Nous définissons une exploration par dépliage de l'espace d'états du modèle stochastique, guidée par le journal. Cette procédure identifie et explore systématiquement les branches du modèle capables de générer les traces observées dans le journal, tout en conservant les probabilités cumulées associées à chaque chemin du modèle.
- *Méthodes approximatives.* Nous proposons une approche approximative fondée sur la simulation de modèles, guidée par des logiques d'automates hybrides. Cette approche consiste à échantillonner un grand nombre d'exécutions du modèle et à les utiliser pour approximer son comportement stochastique.

Ces deux méthodes sont intégrées dans un cadre itératif spécifique de découverte stochastique, adapté à la stratégie retenue.

(Q2) Étant donné un modèle stochastique paramétré, peut-on concevoir une procédure permettant d'identifier les paramètres stochastiques optimaux (au sens d'une mesure de conformité stochastique choisie) correspondant au comportement observé dans le journal ?

Dans une approche indirecte de la découverte de processus stochastiques, on part d'un modèle de flot de contrôle et l'on calibre ses paramètres stochastiques (par exemple les probabilités de branchement ou les poids de transition) de manière à ce que le comportement probabiliste induit par le modèle s'aligne sur la distribution empirique observée dans le journal d'événements. Nous étudions deux stratégies itératives complémentaires :

- *Estimation de paramètres par optimisation.* Nous définissons une fonction objectif permettant de quantifier la divergence entre le modèle et le journal à l'aide de métriques de conformité stochastique, puis utilisons des techniques d'optimisation pour minimiser cette divergence. Des solveurs basés sur la descente de gradient ainsi que des solveurs sans gradient sont considérés.
- *Inférence bayésienne fondée sur la simulation.* Nous adoptons une procédure de calcul bayésien approximatif (ABC-SMC) afin de construire des distributions a posteriori sur les paramètres stochastiques du modèle. Les valeurs des paramètres sont échantillonnées de manière itérative par simulation des exécutions du modèle, et seules celles atteignant un niveau de conformité suffisamment proche du journal sont conservées, selon un schéma de tolérance progressivement décroissant.

Ces deux stratégies offrent des moyens complémentaires de calibrer des modèles de processus stochastiques : les approches fondées sur l'optimisation visent des estimations ponctuelles efficaces des paramètres, tandis que l'inférence bayésienne fournit des distributions a posteriori complètes, au prix d'un coût computationnel plus élevé.

(Q3) Peut-on étendre des modèles de processus hiérarchiques, tels que les arbres de processus, avec une sémantique stochastique capable de capturer fidèlement le comportement probabiliste observé dans les journaux d'événements ?

Au-delà du calibrage de paramètres stochastiques sur un réseau de Petri donné, nous cherchons un formalisme de modélisation alternatif qui soit à la fois découvrable à partir des journaux et exploitable pour l'analyse quantitative. Les réseaux de Petri stochastiques classiques offrent une sémantique rigoureuse, mais souffrent souvent d'ambiguïtés structurelles et de difficultés d'identifiabilité des paramètres lorsqu'ils sont utilisés dans un contexte de découverte. Pour surmonter ces limitations, nous postulons que les *arbres de processus stochastiques* constituent une représentation appropriée. Un arbres de processus stochastiques étend le formalisme bien établi des arbres de processus [49] en y associant des annotations probabilistes sur les opérateurs de flot de contrôle tels que la séquence, le choix, le parallèle et la boucle. Chaque opérateur est muni de paramètres définissant la vraisemblance des chemins alternatifs, la distribution des entrelacements concurrents ou encore le nombre attendu d'itérations des boucles. La sémantique des arbres est compositionnelle, ce qui signifie que le comportement stochastique d'un modèle complexe est entièrement déterminé par le comportement de ses sous-composants, combinés selon leurs opérateurs respectifs. Comparés à d'autres mo-

dèles stochastiques, les arbres réduisent l'ambiguïté structurelle, en évitant l'existence de plusieurs modèles distincts générant le même comportement, et améliorent l'identifiabilité des paramètres en rendant les choix probabilistes explicites dans la syntaxe du modèle. Ils offrent ainsi un cadre unifié dans lequel la structure peut être directement découverte à partir des journaux, les paramètres calibrés, et le modèle résultant analysé à la fois qualitativement et quantitativement.

8.3 Contributions de la thèse

La capacité à prendre en compte la nature probabiliste d'un processus réel est d'une importance primordiale dans le domaine du *process mining*, en particulier lorsque l'analyse quantitative des métriques de performance constitue un élément clé de la validation d'un processus. Si la découverte classique de processus s'est révélée être une réussite notable, donnant lieu à de nombreuses méthodes efficaces [71, 86, 49, 76] capables d'extraire des modèles capturant les aspects de flot de contrôle présents dans les exécutions observées d'un processus, elle montre en revanche des limites intrinsèques dès lors que l'on considère la dimension stochastique du phénomène. Cette insuffisance a conduit à un intérêt croissant de la communauté de recherche pour les extensions stochastiques du problème de découverte de processus, avec plusieurs contributions proposées ces dernières années [21, 22, 17, 45]. Prendre en compte la dimension probabiliste d'un journal revient à dériver un modèle stochastique capable de reproduire de manière fiable non seulement les aspects de flot de contrôle des traces observées, mais également les probabilités avec lesquelles elles ont été observées. Indépendamment du formalisme considéré, les modèles de processus stochastiques sont caractérisés par un vecteur de paramètres qui influence leur comportement stochastique ; plus concrètement, ces paramètres gouvernent la manière dont la masse de probabilité est distribuée sur les mots du langage qu'ils génèrent.

Dans ce contexte, en supposant qu'un modèle de flot de contrôle soit donné, le cœur du problème de la découverte de processus stochastiques consiste à identifier des paramètres optimaux tels que l'extension stochastique correspondante du modèle distribue la masse de probabilité sur les mots qu'elle génère de manière similaire à la distribution de probabilité observée sur les traces du journal. Autrement dit, la découverte de processus stochastiques se ramène à un problème d'optimisation fondé sur une fonction objectif mesurant la similarité entre des distributions de probabilité discrètes, comme par exemple la distance de *Earth Mover's* [52, 53] ou la pertinence entropique [63, 46, 5]. La découverte de paramètres optimaux pour un modèle de processus stochastique est entravée par deux obstacles corrélés qui affectent fortement la complexité de la tâche : la détermination du langage stochastique du modèle et la dimension de l'espace des paramètres (c'est-à-dire le nombre de paramètres du modèle). En effet, plus le modèle est complexe, plus l'espace des paramètres est grand et plus il est difficile d'évaluer le langage qu'il induit.

Dans l'objectif de contribuer aux avancées dans la résolution du problème de la découverte de processus stochastiques, cette thèse présente un ensemble de

contributions visant à traiter les principaux obstacles qui en compromettent la résolution. Plus précisément :

Calcul exact du langage d'un réseau de Petri stochastique. Dans le cadre des réseaux de Petri stochastiques, qui constituent sans doute la classe de modèles la plus utilisée en *process mining*, nous avons introduit une méthode permettant de calculer exactement la probabilité des traces émises par une instance de réseau stochastique (c'est-à-dire un réseau muni d'un vecteur de poids spécifique pour ses transitions). Cette approche repose sur un parcours en largeur du graphe d'atteignabilité du réseau et permet ainsi de déplier exhaustivement toutes les traces générées par le modèle, tout en propageant de manière incrémentale le calcul numérique de leur probabilité au fur et à mesure du dépliage. Afin de garantir la terminaison, la méthode proposée peut être adaptée pour prendre en compte différents critères d'arrêt, par exemple en interrompant le dépliage dès que la masse de probabilité des traces actuellement dépliées atteint une borne inférieure donnée. Dans notre implémentation, supposant que la procédure opère sur des modèles parfaitement conformes, nous avons choisi d'arrêter le dépliage dès que l'ensemble des traces du journal a été exploré.

Ainsi, en pratique, étant donné un journal, un modèle de réseau de Petri parfaitement ajusté dérivé de ce journal et un vecteur de poids spécifique pour les transitions du modèle, nous avons établi un cadre computationnel permettant de calculer la probabilité exacte avec laquelle les traces du journal apparaissent dans le modèle. Il convient de noter que, dans la plupart des cas, la somme des probabilités associées aux traces du journal dans le modèle n'est pas égale à 1 ; une étape de normalisation est donc nécessaire afin d'intégrer ces résultats dans un cadre d'optimisation. Le principal avantage de cette méthode est que, contrairement aux approches visant à dériver des expressions symboliques pour la probabilité des transitions [45], qui souffrent de sérieux problèmes de passage à l'échelle, elle s'est révélée efficace sur des modèles extraits de journaux de référence couramment utilisés.

Afin d'améliorer encore l'efficacité, nous avons ensuite développé une procédure de dépliage enrichie dans laquelle, plutôt que de stocker les valeurs numériques des probabilités associées à chaque trace, nous stockons la fonction (dépendant des paramètres de poids) permettant de calculer cette probabilité pour un vecteur de poids donné. Cela permet d'éviter de répéter le dépliage lors de l'évaluation des probabilités des traces pour différents vecteurs de paramètres. Une fois le graphe d'atteignabilité construit pour un modèle donné, il peut être réutilisé pour extraire le langage induit par n'importe quel vecteur de poids, tout en évitant des calculs redondants grâce à la mémoïsation. Malgré ce gain substantiel en efficacité computationnelle, le principal inconvénient de l'approche par dépliage demeure sa complexité spatiale, car la taille du graphe d'atteignabilité peut exploser en fonction des caractéristiques du journal (et donc de la structure du modèle), ce qui limite l'applicabilité de la méthode à des modèles complexes qui présentent un grand nombre d'états.

Découverte de poids optimaux pour un modèle de réseaux de Petri stochastique. En nous appuyant sur le calcul exact du langage d'un réseau de Petri stochastique, nous avons ensuite défini un cadre permettant de découvrir des poids optimaux pour un modèle donné. À cette fin, nous avons considéré deux types de fonctions objectif. La première est une version personnalisée de la distance de *Earth Mover's*, que nous avons nommée distance de *Earth Mover's* restreinte, car elle ne prend en compte que l'intersection entre les traces du modèle et celles du journal lors du calcul de la quantité minimale de masse de probabilité à déplacer. La seconde est une fonction différentiable, la fonction de log-vraisemblance, dérivée de l'estimation du maximum de vraisemblance, qui évalue la probabilité que le modèle génère le journal d'événements observé.

Le schéma d'optimisation repose à la fois sur des solveurs basés sur la descente de gradient et sur des solveurs sans gradient afin d'explorer l'espace des paramètres et d'affiner le vecteur de poids associé au réseau de Petri stochastique, dans le but de minimiser l'une des métriques considérées. À chaque itération, le solveur propose un nouveau vecteur de poids et la conformité du modèle est évaluée en appliquant la procédure de dépliage.

Inférence des poids optimaux d'un réseau de Petri stochastique par simulation. Afin de dépasser les limites liées au coût potentiel des calculs numériques exacts du langage d'un réseau, nous avons envisagé une alternative fondée sur la simulation. Classiquement, l'intérêt des approches par simulation réside dans la possibilité d'échanger précision contre temps de calcul. Dans cette optique, nous avons introduit une nouvelle méthode reposant sur la vérification statistique de modèles, permettant d'obtenir une approximation arbitrairement précise des probabilités des traces générées par un réseau de Petri stochastique. Cette approximation est ensuite combinée à un schéma d'inférence bayésienne des paramètres, permettant une exploration efficace de l'espace des paramètres guidée par un critère de conformité stochastique donné.

Nous avons validé cette approche au moyen d'expériences menées sur des journaux de référence, montrant que, pour plusieurs journaux, elle surpasse la méthode d'optimisation fondée sur le calcul exact, en identifiant des poids de transition conduisant à une distance de conformité plus faible, pour un temps de calcul comparable.

Réduction de la dimension de l'espace des paramètres par le formalisme des arbres de processus stochastiques. Un aspect critique des modèles de réseaux de Petri découverts à partir d'un journal est que, du fait de la stratégie de découverte employée, ils contiennent souvent de nombreuses transitions silencieuses redondantes. Ces transitions peuvent être considérées comme redondantes dans la mesure où, bien qu'elles jouent un rôle dans la reproduction du flot de contrôle des traces du journal, elles induisent également un nombre potentiellement élevé d'entrelacements pour une même trace générée par le réseau (c'est-à-dire qu'il peut exister plusieurs façons distinctes pour le modèle de produire une même trace). C'est notamment le cas des réseaux générés par l'algorithme *Inductive Miner*, qui

résultent d'une étape de traduction d'un arbre de processus découvert vers un réseau de Petri équivalent.

Pour atténuer cette limitation, nous avons introduit l'extension stochastique des arbres de processus, qui constitue le formalisme source à partir duquel les réseaux de Petri sont dérivés dans l'*Inductive Miner*. Les arbres de processus stochastiques présentent un double avantage. D'une part, ils réduisent le nombre de paramètres dont dépendent les modèles, ce qui est clairement bénéfique pour la recherche de paramètres optimaux, et d'autre part, et plus important encore, ils éliminent l'ambiguïté liée aux transitions silencieuses redondantes des réseaux de Petri.

À partir de la formalisation de la syntaxe et de la sémantique des arbres de processus stochastiques, nous avons défini une procédure de simulation stochastique permettant d'échantillonner des traces générées par un modèle d'arbre. Sur cette base, nous avons développé un moteur fondé sur la simulation pour la découverte de paramètres optimaux d'un modèle d'arbres de processus stochastique.

8.4 Perspectives

Malgré les avancées significatives que représentent les contributions présentées ci-dessus pour la découverte de processus stochastiques, plusieurs extensions et questions ouvertes méritent d'être envisagées dans le cadre de travaux futurs.

Extension de l'évaluation expérimentale. Une direction relativement naturelle pour des travaux futurs consiste à élargir l'évaluation expérimentale de l'ensemble des méthodes proposées à un corpus plus vaste et plus diversifié de journaux d'événements, variant en taille et en complexité. Nous souhaitons également évaluer le comportement de nos approches sur des études de cas réelles spécifiques, ainsi que sur des journaux synthétiques volontairement conçus pour exhiber des comportements atypiques ou difficiles, tels que des boucles profondément imbriquées, des motifs de concurrence complexes ou des traces exceptionnelles rares. De telles expérimentations permettraient d'identifier des limitations qui ne peuvent pas être mises en évidence à partir du jeu de benchmarks actuellement considéré.

Exploration de mesures alternatives de conformité stochastique. Dans le même esprit, nous envisageons d'étendre nos méthodes afin de prendre en charge d'autres mesures de conformité et de distance stochastiques, telles que la pertinence entropique [46], qui s'est imposée comme l'une des approches les plus efficaces et les plus largement adoptées pour quantifier la similarité entre langages stochastiques.

Gestion des transitions silencieuses dans le dépliage des réseaux de Petri stochastique. Tout au long de ce travail, l'un des défis majeurs rencontrés lors de la manipulation des réseaux de Petri concerne la présence de transitions silencieuses. Les inconvénients induits par ces transitions ont d'ailleurs constitué la motivation initiale nous ayant conduit à explorer le formalisme des abres de processus. Dans le

cadre de la procédure de dépliage décrite au Chapitre 3, les transitions silencieuses sont la principale cause de la croissance exponentielle de la taille du graphe d'atteignabilité généré et exploré.

À court terme, nous étudions des améliorations de la procédure de dépliage visant à « supprimer le silence » du graphe d'atteignabilité du réseau. L'idée consiste à dériver, à partir du graphe d'atteignabilité annoté par des probabilités, un graphe réduit, dans lequel les comportements silencieux redondants seraient éliminés. Dans un second temps, nous prévoyons de représenter le graphe d'atteignabilité sous la forme d'une matrice de transition stochastique capturant la vraisemblance des passages entre marquages, puis d'appliquer une décomposition matricielle afin de séparer la contribution de chaque transition. Une telle approche permettrait de mesurer précisément l'influence des transitions silencieuses sur le comportement stochastique global du réseau, et ainsi de les traiter de manière plus efficace.

Exploitation approfondie de la distribution marginale a posteriori issue de l'*ABC-SMC*. La majorité des méthodes indirectes de découverte stochastique sont conçues pour retourner un unique optimum local dans l'espace des paramètres stochastiques, offrant ainsi une vision limitée du paysage global des solutions possibles. À l'inverse, l'approche *ABC-SMC* fournit un résultat bien plus riche, en produisant une approximation de la distribution a posteriori complète des paramètres de poids. À ce stade, nous n'exploitons toutefois qu'une fraction limitée de cette information lors de la découverte.

Cette distribution a posteriori permet non seulement d'identifier des régions de forte probabilité correspondant à des configurations de paramètres quasi optimales, mais également de révéler des corrélations entre les poids ainsi qu'une éventuelle multimodalité de l'espace des solutions. À l'heure actuelle, cette multimodalité potentielle, lorsqu'elle existe, ne peut pas être observée, car la seule loi étudiée pour les poids est une loi normale tronquée classique [16]. Des travaux futurs viseront à exploiter plus finement cette information, par exemple pour mieux comprendre l'impact de chaque transition du réseau, détecter des comportements stochastiques alternatifs compatibles avec le journal observé, ou encore initialiser et guider d'autres procédures d'optimisation.

Calcul exact du langage d'un arbre de processus stochastique. La découverte indirecte d'arbres de processus stochastique repose actuellement sur une procédure d'optimisation fondée sur la simulation. Nous soutenons qu'un tel processus d'optimisation bénéficierait d'un calcul exact du langage stochastique, plutôt que d'une approximation par échantillonnage, car cela fournirait au solveur un guidage plus précis pour identifier des valeurs optimales des paramètres. Nous travaillons donc à la définition d'une procédure de parcours des arbres permettant le calcul exact de leur langage stochastique.

Traduction des arbres de processus stochastique en réseaux de Petri stochastiques. Le résultat actuel de la méthode de découverte stochastique décrite au Cha-

pitre 6 est un arbre de processus stochastique optimisé. Toutefois, nous ne disposons pas encore d'une méthode systématique permettant de traduire la sémantique des opérateurs d'un arbre vers un formalisme plus largement implémenté, tel que les réseaux de Petri stochastique.

Dans un premier temps, nous étudions activement une extension stochastique du mappage entre arbres de processus et réseaux de Petri initialement introduit dans l'article fondateur de l'*Inductive Miner* [49], afin de permettre la génération automatique d'un réseaux stochastique équivalent à partir d'un arbres stochastique donné. Des tentatives préliminaires ont montré que cette traduction est loin d'être triviale, voire potentiellement impossible dans le cas général, car préserver exactement la sémantique stochastique nécessiterait d'établir une correspondance cohérente entre les paramètres probabilistes des nœuds de l'arbre et les paramètres de poids des transitions du réseau. Atteindre une telle équivalence pourrait en outre requérir des adaptations structurelles du réseau résultant afin de reproduire fidèlement la sémantique de l'arbre d'origine.

Dans un second temps, nous travaillons à la traduction formelle du formalisme des abres de processus stochatique vers des langages couramment utilisés en vérification probabiliste de modèles, tels que l'algèbre de processus *PEPA* [37] et le langage des *Reactive Modules* [7]. Ces deux langages sont supportés par le vérificateur de modèles *PRISM* [43] et permettent des spécifications de haut niveau de chaînes de Markov. Une telle traduction ouvrirait la voie à une analyse formelle expressive et rigoureuse des modèles d'arbres à l'aide de propriétés de logique temporelle (stochastique).

8.5 Conclusion

En conclusion, la prise en compte de la stochasticité dans le comportement des processus est essentielle pour étendre le champ d'application du *process mining* au-delà de ses limites actuelles. Elle accroît le pouvoir expressif des modèles et permet une représentation plus fidèle et plus nuancée des dynamiques du monde réel. Dans le même temps, elle conduit naturellement à l'exploration d'un autre aspect fondamental, et encore plus complexe, de la réalité : la dimension temporelle.

Jusqu'à présent, les horodatages présents dans les journaux d'événements ont principalement été exploités dans les approches de découverte structurelle afin de déterminer l'ordre d'exécution des activités au sein des traces. Cependant, cette information temporelle peut être utilisée bien au-delà de la simple notion d'ordonnancement, notamment pour modéliser et analyser les temps d'attente entre les activités, offrant ainsi des perspectives précieuses sur la performance des processus et le comportement des ressources. L'intégration explicite du temps comme dimension quantitative ouvrirait la voie à des analyses prédictives avancées, telles que l'estimation des temps de traitement restants, l'identification de goulets d'étranglement ou encore la prévision de l'issue des processus.

La combinaison des perspectives stochastique et temporelle constitue ainsi une étape prometteuse vers des modèles de *process mining* plus réalistes et plus expressifs. Dans cette optique, et en s'appuyant sur des travaux précurseurs [73], nous envisageons comme perspective future l'étude approfondie de la modélisation explicite du temps au sein des modèles de processus. Cela implique l'extension de formalismes existants, tels que les réseaux de Petri et les arbres de processus, par l'introduction de paramètres temporels, ainsi que l'exploitation de nos méthodes de découverte fondées sur l'optimisation afin d'estimer ces paramètres temporels de manière à ce qu'ils s'ajustent au mieux au journal d'événements, permettant ainsi de capturer une part encore plus riche des dynamiques fondamentales contenues dans les données.

Appendix A

Second-order derivatives of the log-likelihood function

Some optimisation methods can benefit from the use of the so-called Hessian matrix, which, in this context, consists of the second-order partial derivatives of the log-likelihood function with respect to the model parameters. These second derivatives can also be computed during the unfolding by propagating them from the parent nodes to each node in the structure. Since the Hessian matrix is symmetric, only $n(n+1)/2$ distinct second-order derivatives need to be maintained per node. As a result, each node must store its probability, its gradient (first-order derivatives), and its Hessian entries.

In order to obtain the second-order partial derivatives of the log-likelihood function with respect to the weights, we calculate the derivative of (4.5) with respect to w_m (with $1 \leq m \leq n$) as

$$\frac{\partial^2 g(\overline{w})}{\partial w_l \partial w_m} = \sum_{i=1}^{k} \left(\frac{\partial^2 f_i(\overline{w})}{\partial w_l \partial w_m} \cdot \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j} + \frac{\partial f_i(\overline{w})}{\partial w_l} \cdot \frac{\partial \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j}}{\partial w_m} + \frac{\partial f_i(\overline{w})}{\partial w_m} \cdot \frac{\partial \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j}}{\partial w_l} + f_i(\overline{w}) \frac{\partial^2 \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j}}{\partial w_l \partial w_m} \right)$$

where, other than the probability, the derivatives and the partial derivatives of the parent nodes, we have the terms

$$\frac{\partial \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j}}{\partial w_m} \quad \text{and} \quad \frac{\partial \frac{w_{c_i}}{\sum_{j \in \mathcal{D}_i} w_j}}{\partial w_l}$$

given in (4.6), and

$$\frac{\partial^2 \frac{w_{c_i}}{\sum_{j\in\mathcal{D}_i} w_j}}{\partial w_l \partial w_m} = \begin{cases} 0 & \text{if } l \notin \mathcal{D}_i \vee m \notin \mathcal{D}_i \\ \dfrac{2w_{c_i}}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^3} & \text{if } l \in \mathcal{D}_i \wedge l \neq c_i \wedge m \in \mathcal{D}_i \wedge m \neq c_i \\ \dfrac{w_l - \sum_{j\in\mathcal{D}_i \wedge j\neq l} w_j}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^3} & \text{if } l \in \mathcal{D}_i \wedge l = c_i \wedge m \in \mathcal{D}_i \wedge m \neq c_i \\ \dfrac{w_m - \sum_{j\in\mathcal{D}_i \wedge j\neq m} w_j}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^3} & \text{if } l \in \mathcal{D}_i \wedge l \neq c_i \wedge m \in \mathcal{D}_i \wedge m = c_i \\ -\dfrac{2\sum_{j\in\mathcal{D}_i \wedge j\neq m} w_j}{\left(\sum_{j\in\mathcal{D}_i} w_j\right)^3} & \text{if } l \in \mathcal{D}_i \wedge l = c_i \wedge m \in \mathcal{D}_i \wedge m = c_i \end{cases}$$

Appendix B

Additional Optimisation Consistency Experiments

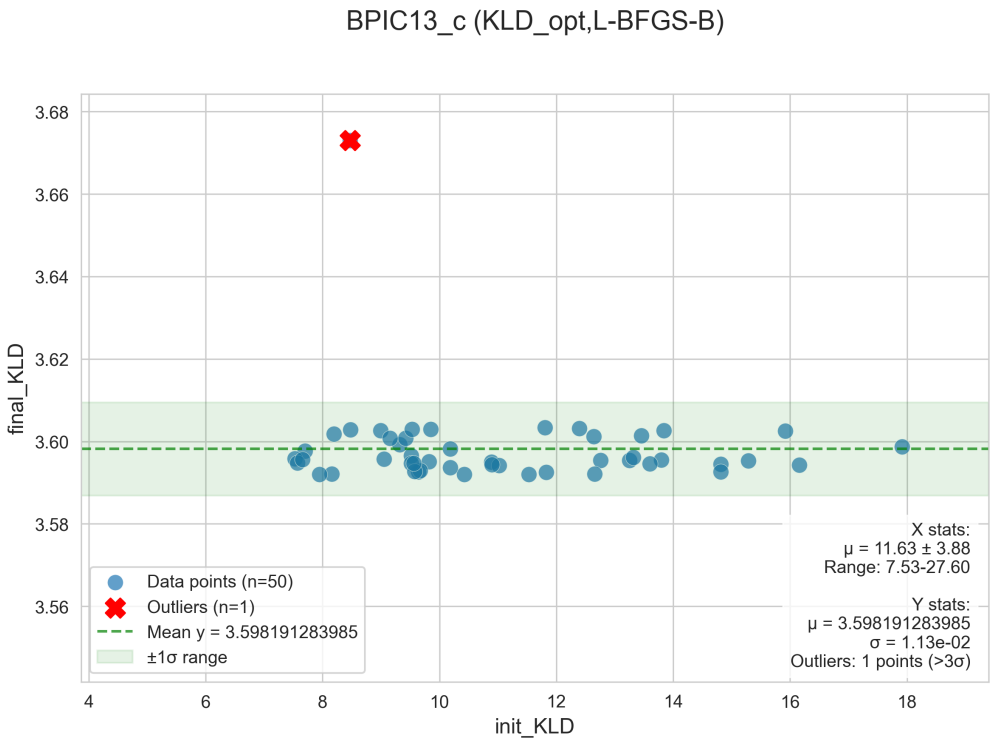


Figure B.1 – Consistency of LH-driven optimisation for the `BPIC13_c` log

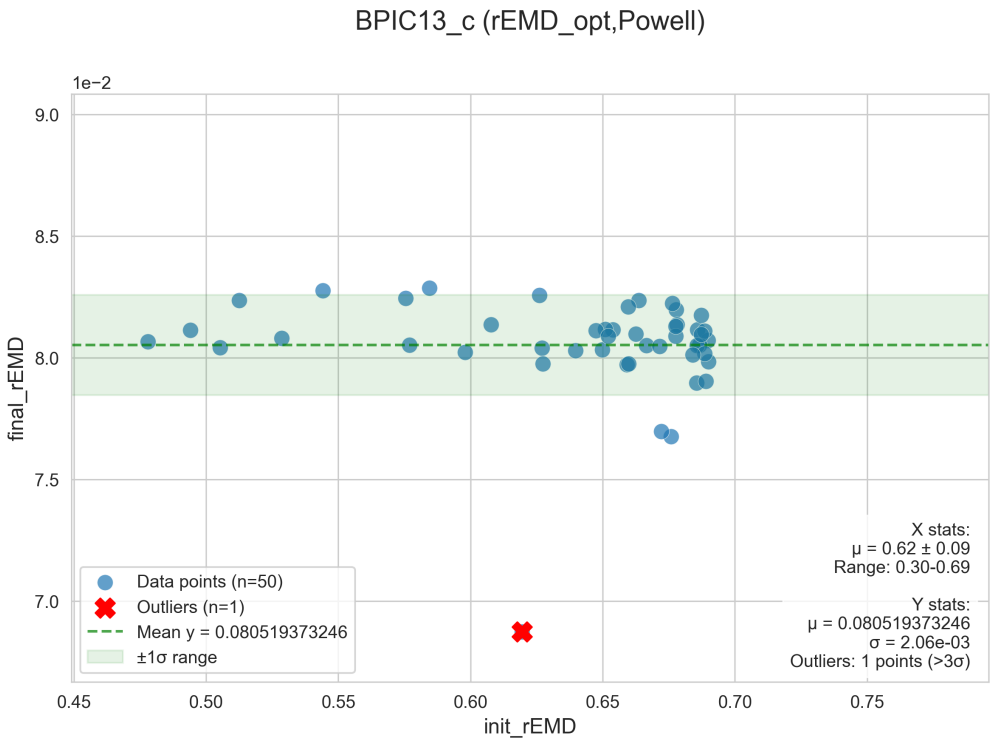


Figure B.2 – Consistency of rEMD-driven optimisation for the `BPIC13_c` log

Bibliographie

[1] Van Der Aalst and W. M. P. Business Process Management. In *Encyclopedia of Database Systems*, pages 370–374. Springer, New York, NY, 2018.

[2] van der Aalst, W.M.P. *Process mining : discovery, conformance and enhancement of business processes*. Springer, Berlin, 2011.

[3] Wil Aalst. *Academic View : Development of the Process Mining Discipline*, pages 181–196. 03 2020.

[4] Arya Adriansyah, B.F. Dongen, and Wil Aalst. *Conformance Checking Using Cost-Based Fitness Analysis*. October 2011. Journal Abbreviation : The Journal of Physical Chemistry Pages : 64, Publication Title : The Journal of Physical Chemistry.

[5] Hanan Alkhammash, Artem Polyvyanyy, Alistair Moffat, and Luciano García-Bañuelos. Entropic relevance : A mechanism for measuring stochastic process models discovered from event data. *Information Systems*, 107 :101922, July 2022.

[6] Anti Alman, Fabrizio Maria Maggi, Marco Montali, and Rafael Peñaloza. Probabilistic declarative process mining. *Information Systems*, 109 :102033, November 2022.

[7] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1) :7–48, 1999.

[8] Adriano Augusto, Raffaele Conforti, Marlon Dumas, and Marcello La Rosa. Split miner : Discovering accurate and simple business process models from event logs. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1–10, 2017.

[9] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs : Review and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 31(4) :686–705, 2019.

[10] Paolo Ballarini, Benoît Barbot, Marie Duflot, Serge Haddad, and Nihal Pekergin. HASL : A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90 :53–77, August 2015.

[11] Paolo Ballarini, Andras Horvath, and Pierre Cry. Probabilistic Process Discovery with Stochastic Process Trees. In *Conference on Perfomance Evaluation and Optimization of Complex Systems*, Milan, Italy, December 2024.

[12] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1) :164–171, 1970.

[13] Mark A. Beaumont, Jean-Marie Cornuet, Jean-Michel Marin, and Christian P. Robert. Adaptive approximate bayesian computation. *Biometrika*, 96(4) :983–990, 2009.

[14] Alessandro Berti, Sebastiaan J Van Zelst, and Wil van der Aalst. Process mining for python (pm4py) : bridging the gap between process-and data science. *arXiv preprint arXiv :1905.06169*, 2019.

[15] Fábio Bezerra, Jacques Wainer, and W. M. P. van der Aalst. Anomaly detection using process mining. In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 149–161, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[16] Z. I. Botev. The normal law under linear restrictions : Simulation and estimation via minimax tilting. *Journal of the Royal Statistical Society Series B : Statistical Methodology*, 79(1) :125–148, 02 2016.

[17] Tobias Brockhoff, Merih Seran Uysal, and Wil Aalst. Wasserstein weight estimation for stochastic petri nets. pages 81–88, 10 2024.

[18] J.C.A.M. Buijs, B.F. {Dongen, van}, and W.M.P. {Aalst, van der}. On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman, editor, *On the Move to Meaningful Internet Systems : OTM 2012 (Confederated International Conferences : CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I)*, Lecture Notes in Computer Science, pages 305–322, Germany, 2013. Springer. 20th International Conference on Cooperative Information Systems, CoopIS 2012, CoopIs 2012 ; Conference date : 12-09-2012 Through 14-09-2012.

[19] Adam Burke. *Process mining with labelled stochastic nets*. PhD thesis, Queensland University of Technology, 2024.

[20] Adam Burke, Sander Leemans, and Moe Wynn. Report on stochastic process discovery by weight estimation experimental results., 2020.

[21] Adam Burke, Sander Leemans, and Moe Wynn. Stochastic process discovery by weight estimation. 10 2020.

[22] Adam Burke, Sander J. J. Leemans, and Moe Thandar Wynn. Discovering stochastic process models by reduction and abstraction. In Didier Buchs and Josep Carmona, editors, *Application and Theory of Petri Nets and Concurrency*, pages 312–336, Cham, 2021. Springer International Publishing.

[23] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5) :1190–1208, 1995.

[24] Josep Carmona, Boudewijn Van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking : Relating Processes and Models*. Springer International Publishing, Cham, 2018.

[25] Ramon C. Carrasco and Jose Oncina. Stochastic inference of regular languages from positive data. In *Proceedings of the International Colloquium on Grammatical Inference (ICGI 1994)*, pages 139–152. Springer, 1994.

[26] G. Chiola, M.A. Marsan, G. Balbo, and G. Conte. Generalized stochastic petri nets : a definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2) :89–107, 1993.

[27] Claudio Di Ciccio and Massimo Mecella. On the Discovery of Declarative Control Flows for Artful Processes. *ACM Trans. Manage. Inf. Syst.*, 5(4) :24 :1–24 :37, January 2015.

[28] Pierre Cry, Paolo Ballarini, András Horváth, and Pascale Le Gall. Statistical Bayesian Inference for Stochastic Process Discovery. In *Proceedings of the International Conference on Quantitative Evaluation of SysTems (QEST)*, Aarhus (Danemark), Denmark, August 2025.

[29] Pierre Cry, András Horváth, Paolo Ballarini, and Pascale Le Gall. A framework for optimisation based stochastic process discovery. In Jane Hillston, Sadegh Soudjani, and Masaki Waga, editors, *Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems*, pages 34–51, Cham, 2024. Springer Nature Switzerland.

[30] M. (Massimiliano) de Leoni and Felix Mannhardt. Road Traffic Fine Management Process, February 2015.

[31] Karl Doerner, Walter J. Gutjahr, Gabriele Kotsis, Martin Polaschek, and Christine Strauss. Enriched workflow modelling and Stochastic Branch-and-Bound. *European Journal of Operational Research*, 175(3) :1798–1817, December 2006.

[32] Olga Kouchnarenko Fabien Dagnat. Actes des journées approches formelles dans l'assistance au developpement du logiciel (afadl 2025). In *Approches Formelles dans l'Assistance au Developpement du Logiciel*, June 2025.

[33] Fuchang Gao and Lixing Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51 :259–277, 05 2012.

[34] Christian W Gunther and HMW Verbeek. Xes-standard definition. 2014.

[35] Kees Hee, Hajo Reijers, Eric Verbeek, and Loucif Zerguini. On the optimal allocation of resources. 08 2001.

[36] Thomas A Henzinger. The theory of hybrid automata. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996.

[37] Jane Hillston. A compositional approach to performance modelling. *Distinguished Dissertations in Computer Science*, 1996.

[38] Tsung-Hao Huang and Wil M. P. van der Aalst. Unblocking inductive miner. In Han van der Aa, Dominik Bork, Henderik A. Proper, and Rainer Schmidt, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 327–342, Cham, 2023. Springer Nature Switzerland.

[39] Gert Janssenswillen, Benoît Depaire, and Christel Faes. *Enhancing Discovered Process Models Using Bayesian Inference and MCMC*, pages 295–307. 01 2020.

[40] Agnes Koschmider, Kay Kaczmarek, Mathias Krause, and Sebastiaan van Zelst. *Demystifying Noise and Outliers in Event Logs : Review and Future Directions*, pages 123–135. 01 2022.

[41] Agnes Koschmider, Andreas Oberweis, and Wolffried Stucky. A Petri net-based View on the Business Process Life-Cycle. *Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling*, 13 :47–55, February 2018.

[42] Solomon Kullback. *Information Theory and Statistics*. Wiley, 1959. Google-Books-ID : XeRQAAAAMAAJ.

[43] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0 : Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV 2011)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[44] R.G. Lee and B.G. Dale. Business process management : a review and evaluation. *Business Process Management Journal*, 4(3) :214–225, September 1998.

[45] Sander Leemans, Tian Li, Marco Montali, and Artem Polyvyanyy. *Stochastic Process Discovery : Can It Be Done Optimally ?*, pages 36–52. 06 2024.

[46] Sander Leemans and Artem Polyvyanyy. *Stochastic-Aware Conformance Checking : An Entropy-Based Approach*, pages 217–233. 06 2020.

[47] Sander Leemans, Erik Poppe, and Moe Wynn. Directly follows-based process mining : A tool. In *Proceedings of the ICPM Demo Track 2019 (CEUR Workshop Proceedings, Volume 2374). Vol. 2374.*, pages 9–12. Sun SITE Central Europe, 2019.

[48] Sander J. J. Leemans. Leveraging frequencies in event data : a pledge for stochastic process mining. 2022.

[49] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In José-Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency*, pages 311–329, Berlin, Heidelberg, 2013. Springer.

[50] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, pages 66–78, Cham, 2014. Springer International Publishing.

[51] Sander J. J. Leemans, Fabrizio Maria Maggi, and Marco Montali. Enjoy the silence : Analysis of stochastic Petri nets with silent transitions. *Information Systems*, 124 :102383, September 2024.

[52] Sander J. J. Leemans, Anja F. Syring, and Wil M. P. van der Aalst. Earth Movers' Stochastic Conformance Checking. In Thomas Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling, editors, *Business Process Management Forum*, pages 127–143, Cham, 2019. Springer International Publishing.

[53] Sander J. J. Leemans, Wil M. P. van der Aalst, Tobias Brockhoff, and Artem Polyvyanyy. Stochastic process mining : Earth movers' stochastic conformance. *Information Systems*, 102 :101724, December 2021.

[54] Sander J.J. Leemans, Erik Poppe, and Moe T. Wynn. Directly follows-based process mining. In *2019 International Conference on Process Mining (ICPM)*, pages 25–32, 2019.

[55] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 1965.

[56] Fabrizio Maggi, Arjan Mooij, and Wil Aalst. *User-Guided Discovery of Declarative Process Models*. March 2011. Journal Abbreviation : Chromatographia Publication Title : Chromatographia.

[57] Dennis GJC Maneschijn, Rob Henk Bemthuis, Jeewanie Jayasinghe Arachchige, Faiza Allah Bukhsh, and Maria E Iacob. Balancing simplicity and complexity in modeling mined business processes : A user perspective. In *International Conference on Enterprise Information Systems*, pages 3–21. Springer, 2022.

[58] Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6) :1167–1180, Nov 2012.

[59] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modelling with generalized stochastic petri nets. *SIGMETRICS Perform. Eval. Rev.*, 26(2) :2, August 1998.

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

[61] Stephen G. Nash. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4) :770–788, 1984.

[62] Michel Petitjean. From Shape Similarity to Shape Complementarity : Toward a Docking Theory. *Journal of Mathematical Chemistry*, 35(3) :147–158, March 2004.

[63] Artem Polyvyanyy, Alistair Moffat, and Luciano García-Bañuelos. An entropic relevance measure for stochastic conformance checking in process mining. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 97–104, 2020.

[64] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2) :155–162, 01 1964.

[65] Wolfgang Reisig. *Petri Nets*. Springer, Berlin, Heidelberg, 1985.

[66] Daniel Reißner, Abel Armas-Cervantes, and Marcello La Rosa. Generalization in automated process discovery : A framework based on event log patterns, 2022.

[67] Andreas Rogge-Solti, Wil M. P. van der Aalst, and Mathias Weske. Discovering stochastic petri nets with arbitrary delay distributions from event logs. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, pages 15–27, Cham, 2014. Springer International Publishing.

[68] John Saldhana and Sol Shatz. UML Diagrams to Object Petri Net Models : An Approach for Modeling and Analysis. April 2000.

[69] Chihiro Shibata and Ryo Yoshinaka. Marginalizing out transition probabilities for several subclasses of pfas. In *Proceedings of the Eleventh International Conference on Grammatical Inference (ICGI 2012)*, volume 21, pages 259–263. PMLR, 2012.

[70] Scott A. Sisson, Yanan Fan, and Mark Beaumont, editors. *Handbook of Approximate Bayesian Computation*. Chapman and Hall/CRC, New York, September 2018.

[71] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining : discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9) :1128–1142, September 2004.

[72] W. M. P. van der Aalst. Verification of workflow nets. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, pages 407–426, Berlin, Heidelberg, 1997. Springer.

[73] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2) :450–475, April 2011.

[74] Wil Van Der Aalst. *Process Mining*. Springer, Berlin, Heidelberg, 2016.

[75] Wil Van Der Aalst, Arya Adriansyah, and Boudewijn Van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews-Data Mining and Knowledge Discovery*, 2(2) :182–192, March 2012. Publisher : Wiley.

[76] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. In Kees M. van Hee and Rüdiger Valk, editors, *Applications and Theory of Petri Nets*, pages 368–387, Berlin, Heidelberg, 2008. Springer.

[77] Boudewijn van Dongen. Bpi challenge 2012, 2012.

[78] Boudewijn van Dongen. BPI Challenge 2020 : Domestic Declarations, March 2020.

[79] Boudewijn van Dongen. BPI Challenge 2020 : Request For Payment, March 2020.

[80] Boudewijn van Dongen. BPI Challenge 2017 - Offer log, February 2021.

[81] Kees van Hee, Natalia Sidorova, and Marc Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In Wil M. P. van der Aalst and Eike Best, editors, *Applications and Theory of Petri Nets 2003*, pages 337–356, Berlin, Heidelberg, 2003. Springer.

[82] Sicco Verwer, Rémi Eyraud, and Colin de la Higuera. Pautomac : a probabilistic automata and hidden markov models learning competition. *Machine Learning*, 96(1) :129–154, 2014.

[83] Ward Steeman. BPI Challenge 2013, closed problems, April 2013.

[84] Ward Steeman. BPI Challenge 2013, incidents, April 2013.

[85] Ward Steeman. BPI Challenge 2013, open problems, April 2013.

[86] A.J.M.M. Weijters, W.M.P. {Aalst, van der}, and A.K. {Alves De Medeiros}. *Process mining with the HeuristicsMiner algorithm*. BETA publicatie : working papers. Technische Universiteit Eindhoven, 2006.

[87] Stephen White. Process Modeling Notations and Workflow Patterns. 2004.